$\Pi-$nets: Deep Polynomial Neural Networks

Grigorios G. Chrysos[1], Stylianos Moschoglou[1,2], Giorgos Bouritsas[1],
Yannis Panagakis[3], Jiankang Deng[1,2], Stefanos Zafeiriou[1,2]
[1] Department of Computing, Imperial College London, UK
[2] Facesoft.io
[3] Department of Informatics and Telecommunications , University of Athens, GR
{[first letter].[surname]}@imperial.ac.uk

Abstract

Deep Convolutional Neural Networks (DCNNs) is currently the method of choice both for generative, as well as for discriminative learning in computer vision and machine learning. The success of DCNNs can be attributed to the careful selection of their building blocks (e.g., residual blocks, rectifiers, sophisticated normalization schemes, to mention but a few). In this paper, we propose Π*-Nets, a new class of DCNNs.* Π*-Nets are polynomial neural networks, i.e., the output is a high-order polynomial of the input.* Π*-Nets can be implemented using special kind of skip connections and their parameters can be represented via high-order tensors. We empirically demonstrate that* Π*-Nets have better representation power than standard DCNNs and they even produce good results* ***without the use of non-linear activation functions*** *in a large battery of tasks and signals, i.e., images, graphs, and audio. When used in conjunction with activation functions,* Π*-Nets produce state-of-the-art results in challenging tasks, such as image generation. Lastly, our framework elucidates why recent generative models, such as StyleGAN, improve upon their predecessors, e.g., ProGAN.*

1. Introduction

Representation learning via the use of (deep) multi-layered non-linear models has revolutionised the field of computer vision the past decade [32, 17]. Deep Convolutional Neural Networks (DCNNs) [33, 32] have been the dominant class of models. Typically, a DCNN is a sequence of layers where the output of each layer is fed first to a convolutional operator (i.e., a set of shared weights applied via the convolution operator) and then to a non-linear activation function. Skip connections between various layers allow deeper representations and improve the gradient flow while training the network [17, 54].

In the aforementioned case, if the non-linear activation functions are removed, the output of a DCNN degenerates to a linear function of the input. In this paper, we propose a new class of DCNNs, which we coin $\Pi-$nets, where the output is a polynomial function of the input. We design $\Pi-$nets for generative tasks (e.g., where the input is a small dimensional noise vector) as well as for discriminative tasks (e.g., where the input is an image and the output is a vector with dimensions equal to the number of labels). We demonstrate that these networks can produce good results without the use of non-linear activation functions. Furthermore, our extensive experiments show, empirically, that $\Pi-$nets can consistently improve the performance, in both generative and discriminative tasks, using, in many cases, significantly fewer parameters.

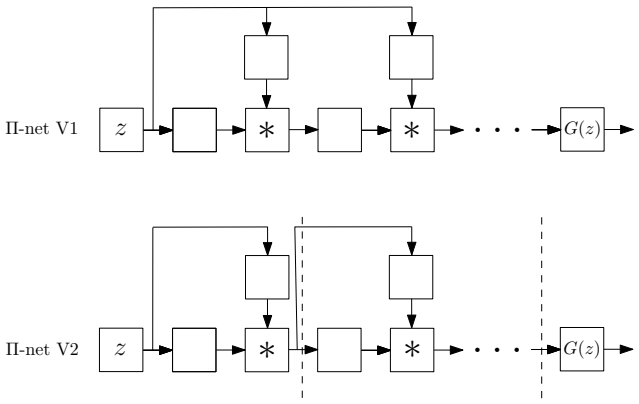

Figure 1: In this paper we introduce a class of networks called $\Pi-$nets, where the output is a polynomial of the input. The input in this case, $\boldsymbol{z}$, can be either the latent space of Generative Adversarial Network for a generative task or an image in the case of a discriminative task. Our polynomial networks can be easily implemented using a special kind of skip connections.

DCNNs have been used in computer vision for over 30 years [33, 50]. Arguably, what brought DCNNs again in mainstream research was the remarkable results achieved by the so-called AlexNet in the ImageNet challenge [32]. Even

though it is only seven years from this pioneering effort the field has witnessed dramatic improvement in all data-dependent tasks, such as object detection [21] and image generation [37, 15], just to name a few examples. The improvement is mainly attributed to carefully selected units in the architectural pipeline of DCNNs, such as blocks with skip connections [17], sophisticated normalization schemes (e.g., batch normalisation [23]), as well as the use of efficient gradient-based optimization techniques [28].

Parallel to the development of DCNN architectures for discriminative tasks, such as classification, the notion of Generative Adversarial Networks (GANs) was introduced for training generative models. GANs became instantly a popular line of research but it was only after the careful design of DCNN pipelines and training strategies that GANs were able to produce realistic images [26, 2]. ProGAN [25] was the first architecture to synthesize realistic facial images by a DCNN. StyleGAN [26] is a follow-up work that improved ProGAN. The main addition of StyleGAN was a type of skip connections, called ADAIN [22], which allowed the latent representation to be infused in all different layers of the generator. Similar infusions were introduced in [42] for conditional image generation.

Our work is motivated by the improvement of StyleGAN over ProGAN by such a simple infusion layer and the need to provide an explanation[1]. We show that such infusion layers create a special non-linear structure, i.e., a higher-order polynomial, which empirically improves the representation power of DCNNs. We show that this infusion layer can be generalized (e.g. see Fig. 1) and applied in various ways in both generative, as well as discriminative architectures. In particular, the paper bears the following contributions:

- We propose a new family of neural networks (called $\Pi-$nets) where the output is a high-order polynomial of the input. To avoid the combinatorial explosion in the number of parameters of polynomial activation functions [27] our $\Pi-$nets use a special kind of skip connections to implement the polynomial expansion (please see Fig. 1 for a brief schematic representation). We theoretically demonstrate that these kind of skip connections relate to special forms of tensor decompositions.

- We show how the proposed architectures can be applied in generative models such as GANs, as well as discriminative networks. We showcase that the resulting architectures can be used to learn high-dimensional distributions without non-linear activation functions.

- We convert state-of-the-art baselines using the proposed $\Pi-$nets and show how they can largely improve the expressivity of the baseline. We demonstrate it conclusively in a battery of tasks (i.e., generation and classification). Finally, we demonstrate that our architectures are applicable to many different signals such as images, meshes, and audio.

[1]The authors argued that this infusion layer is a kind of a style that allows a coarser to finer manipulation of the generation process. We, instead, attribute this to gradually increasing the power of the polynomial.

2. Related work

Expressivity of (deep) neural networks: The last few years, (deep) neural networks have been applied to a wide range of applications with impressive results. The performance boost can be attributed to a host of factors including: a) the availability of massive datasets [4, 35], b) the machine learning libraries [57, 43] running on massively parallel hardware, c) training improvements. The training improvements include a) optimizer improvement [28, 46], b) augmented capacity of the network [53], c) regularization tricks [11, 49, 23, 58]. However, the paradigm for each layer remains largely unchanged for several decades: each layer is composed of a linear transformation and an element-wise activation function. Despite the variety of linear transformations [9, 33, 32] and activation functions [44, 39] being used, the effort to extend this paradigm has not drawn much attention to date.

Recently, hierarchical models have exhibited stellar performance in learning expressive generative models [2, 26, 70]. For instance, the recent BigGAN [2] performs a hierarchical composition through skip connections from the noise $\boldsymbol{z}$ to multiple resolutions of the generator. A similar idea emerged in StyleGAN [26], which is an improvement over the Progressive Growing of GANs (ProGAN) [25]. As ProGAN, StyleGAN is a highly-engineered network that achieves compelling results on synthesized 2D images. In order to provide an explanation on the improvements of StyleGAN over ProGAN, the authors adopt arguments from the style transfer literature [22]. We believe that these improvements can be better explained under the light of our proposed polynomial function approximation. Despite the hierarchical composition proposed in these works, we present an intuitive and mathematically elaborate method to achieve a more precise approximation with a polynomial expansion. We also demonstrate that such a polynomial expansion can be used in both image generation (as in [26, 2]), image classification, and graph representation learning.

Polynomial networks: Polynomial relationships have been investigated in two specific categories of networks: a) self-organizing networks with hard-coded feature selection, b) pi-sigma networks.

The idea of learnable polynomial features can be traced back to Group Method of Data Handling (GMDH) [24][2]. GMDH learns partial descriptors that capture quadratic correlations between two predefined input elements. In [41],

[2]This is often referred to as the first deep neural network [50].

more input elements are allowed, while higher-order polynomials are used. The input to each partial descriptor is predefined (subset of the input elements), which does not allow the method to scale to high-dimensional data with complex correlations.

Shin *et al*. [51] introduce the pi-sigma network, which is a neural network with a single hidden layer. Multiple affine transformations of the data are learned; a product unit multiplies all the features to obtain the output. Improvements in the pi-sigma network include regularization for training in [66] or using multiple product units to obtain the output in [61]. The pi-sigma network is extended in sigma-pi-sigma neural network (SPSNN) [34]. The idea of SPSNN relies on summing different pi-sigma networks to obtain each output. SPSNN also uses a predefined basis (overlapping rectangular pulses) on each pi-sigma sub-network to filter the input features. Even though such networks use polynomial features or products, they do not scale well in high-dimensional signals. In addition, their experimental evaluation is conducted only on signals with known ground-truth distributions (and with up to 3 dimensional input/output), unlike the modern generative models where only a finite number of samples from high-dimensional ground-truth distributions is available.

3. Method

Notation: Tensors are symbolized by calligraphic letters, e.g., $\boldsymbol{\mathcal{X}}$, while matrices (vectors) are denoted by uppercase (lowercase) boldface letters e.g., $\boldsymbol{X}$, ($\boldsymbol{x}$). The *mode-m vector product* of $\boldsymbol{\mathcal{X}}$ with a vector $\boldsymbol{u} \in \mathbb{R}^{I_m}$ is denoted by $\boldsymbol{\mathcal{X}} \times_m \boldsymbol{u}$.[3]

We want to learn a function approximator where each element of the output x_j, with $j \in [1, o]$, is expressed as a polynomial[4] of all the input elements z_i, with $i \in [1, d]$. That is, we want to learn a function $G : \mathbb{R}^d \rightarrow \mathbb{R}^o$ of order $N \in \mathbb{N}$, such that:

$$x_j = G(\boldsymbol{z})_j = \beta_j + \boldsymbol{w}_j^{[1]^T} \boldsymbol{z} + \boldsymbol{z}^T \boldsymbol{W}_j^{[2]} \boldsymbol{z} + \boldsymbol{\mathcal{W}}_j^{[3]} \times_1 \boldsymbol{z} \times_2 \boldsymbol{z} \times_3 \boldsymbol{z} + \cdots + \boldsymbol{\mathcal{W}}_j^{[N]} \prod_{n=1}^{N} \times_n \boldsymbol{z} \tag{1}$$

where $\beta_j \in \mathbb{R}$, and $\left\{\boldsymbol{\mathcal{W}}_j^{[n]} \in \mathbb{R}^{\prod_{m=1}^{n} \times_m d}\right\}_{n=1}^{N}$ are parameters for approximating the output x_j. The correlations (of the input elements z_i) up to N^{th} order emerge in (1). A more compact expression of (1) is obtained by vectorizing the outputs:

$$\boldsymbol{x} = G(\boldsymbol{z}) = \sum_{n=1}^{N} \left(\boldsymbol{\mathcal{W}}^{[n]} \prod_{j=2}^{n+1} \times_j \boldsymbol{z} \right) + \boldsymbol{\beta} \tag{2}$$

[3] A detailed tensor notation is provided in the supplementary.

[4] The theorem of [55] guarantees that any smooth function can be approximated by a polynomial. The approximation of multivariate functions is covered by an extension of the Weierstrass theorem, e.g. in [40] (pg 19).

where $\boldsymbol{\beta} \in \mathbb{R}^o$ and $\left\{\boldsymbol{\mathcal{W}}^{[n]} \in \mathbb{R}^{o \times \prod_{m=1}^{n} \times_m d}\right\}_{n=1}^{N}$ are the learnable parameters. This form of (2) allows us to approximate any smooth function (for large N), however the parameters grow with $\mathcal{O}(d^N)$.

A variety of methods, such as pruning [8, 16], tensor decompositions [29, 52], special linear operators [6] with reduced parameters, parameter sharing/prediction [67, 5], can be employed to reduce the parameters. In contrast to the heuristic approaches of pruning or prediction, we describe below two principled ways which allow an efficient implementation. The first method relies on performing an off-the-shelf tensor decomposition on (2), while the second considers the final polynomial as the product of lower-degree polynomials.

The tensor decompositions are used in this paper to provide a theoretical understanding (i.e., what is the order of the polynomial used) of the proposed family of Π-nets. Implementation-wise the incorporation of different Π-net structures is as simple as the incorporatation of a skip-connection. Nevertheless, in Π-net different skip connections lead to different kinds of polynomial networks.

3.1. Single polynomial

A tensor decomposition on the parameters is a natural way to reduce the parameters and to implement (2) with a neural network. Below, we demonstrate how three such decompositions result in novel architectures for a neural network training. The main symbols are summarized in Table 1, while the equivalence between the recursive relationship and the polynomial is analyzed in the supplementary.

Model 1: CCP: A coupled CP decomposition [29] is applied on the parameter tensors. That is, each parameter tensor, i.e. $\boldsymbol{\mathcal{W}}^{[n]}$ for $n \in [1, N]$, is not factorized individually, but rather a coupled factorization of the parameters is defined. The recursive relationship is:

$$\boldsymbol{x}_n = \left(\boldsymbol{U}_{[n]}^T \boldsymbol{z} \right) * \boldsymbol{x}_{n-1} + \boldsymbol{x}_{n-1} \tag{3}$$

for $n = 2, \ldots, N$ with $\boldsymbol{x}_1 = \boldsymbol{U}_{[1]}^T \boldsymbol{z}$ and $\boldsymbol{x} = \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{\beta}$. The parameters $\boldsymbol{C} \in \mathbb{R}^{o \times k}, \boldsymbol{U}_{[n]} \in \mathbb{R}^{d \times k}$ for $n = 1, \ldots, N$ are learnable. To avoid overloading the diagram, a schematic assuming a third order expansion ($N = 3$) is illustrated in Fig. 2.

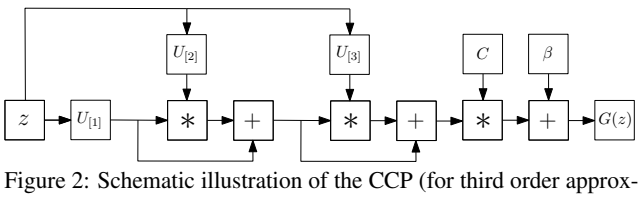

Figure 2: Schematic illustration of the CCP (for third order approximation). Symbol $*$ refers to the Hadamard product.

Model 2: NCP: Instead of defining a flat CP decomposition, we can utilize a joint hierarchical decomposition on the

Table 1: Nomenclature

Symbol	Dimension(s)	Definition
n, N	$\mathbb{N}$	Polynomial term order, total approximation order.
k	$\mathbb{N}$	Rank of the decompositions.
$\boldsymbol{z}$	$\mathbb{R}^d$	Input to the polynomial approximator, i.e., generator.
$\boldsymbol{C}, \boldsymbol{\beta}$	$\mathbb{R}^{o\times k}, \mathbb{R}^o$	Parameters in all decompositions.
$\boldsymbol{A}_{[n]}, \boldsymbol{S}_{[n]}, \boldsymbol{B}_{[n]}$	$\mathbb{R}^{d\times k}, \mathbb{R}^{k\times k}, \mathbb{R}^{\omega\times k}$	Matrix parameters in the hierarchical decomposition.
$\odot, *$	-	Khatri-Rao product, Hadamard product.

polynomial parameters. A nested coupled CP decomposition (NCP), which results in the following recursive relationship for N^{th} order approximation is defined:

$$\boldsymbol{x}_n = \left(\boldsymbol{A}_{[n]}^T \boldsymbol{z}\right) * \left(\boldsymbol{S}_{[n]}^T \boldsymbol{x}_{n-1} + \boldsymbol{B}_{[n]}^T \boldsymbol{b}_{[n]}\right) \tag{4}$$

for $n = 2, \ldots, N$ with $\boldsymbol{x}_1 = \left(\boldsymbol{A}_{[n]}^T \boldsymbol{z}\right) * \left(\boldsymbol{B}_{[n]}^T \boldsymbol{b}_{[n]}\right)$ and $\boldsymbol{x} = \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{\beta}$. The parameters $\boldsymbol{C} \in \mathbb{R}^{o\times k}, \boldsymbol{A}_{[n]} \in \mathbb{R}^{d\times k}, \boldsymbol{S}_{[n]} \in \mathbb{R}^{k\times k}, \boldsymbol{B}_{[n]} \in \mathbb{R}^{\omega\times k}, \boldsymbol{b}_{[n]} \in \mathbb{R}^{\omega}$ for $n = 1, \ldots, N$, are learnable. The explanation of each variable is elaborated in the supplementary, where the decomposition is derived.

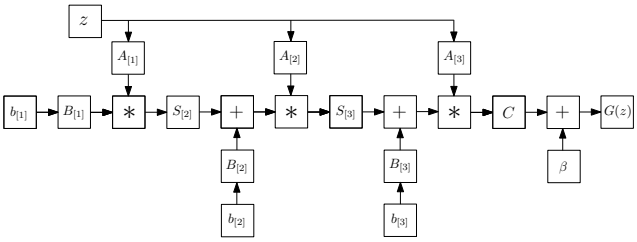

Figure 3: Schematic illustration of the NCP (for third order approximation). Symbol $*$ refers to the Hadamard product.

Model 3: NCP-Skip: The expressiveness of NCP can be further extended using a skip connection (motivated by CCP). The new model uses a nested coupled decomposition and has the following recursive expression:

$$\boldsymbol{x}_n = \left(\boldsymbol{A}_{[n]}^T \boldsymbol{z}\right) * \left(\boldsymbol{S}_{[n]}^T \boldsymbol{x}_{n-1} + \boldsymbol{B}_{[n]}^T \boldsymbol{b}_{[n]}\right) + \boldsymbol{x}_{n-1} \tag{5}$$

for $n = 2, \ldots, N$ with $\boldsymbol{x}_1 = \left(\boldsymbol{A}_{[n]}^T \boldsymbol{z}\right) * \left(\boldsymbol{B}_{[n]}^T \boldsymbol{b}_{[n]}\right)$ and $\boldsymbol{x} = \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{\beta}$. The learnable parameters are the same as in NCP, however the difference in the recursive form results in a different polynomial expansion and thus architecture.

Comparison between the models: All three models are based on a polynomial expansion, however their recursive forms and employed decompositions differ. The CCP has a simpler expression, however the NCP and the NCP-Skip relate to standard architectures using hierarchical composition that have recently yielded promising results in both generative and discriminative tasks. In the remainder of the paper, for comparison purposes we use the NCP by default for the image generation and NCP-Skip for the image classification. In our preliminary experiments, CCP and NCP share a similar performance based on the setting of Sec. 4. In all cases, to mitigate stability issues that might emerge during training, we employ certain normalization schemes that constrain the magnitude of the gradients. An in-depth theoretical analysis of the architectures is deferred to a future version of our work.

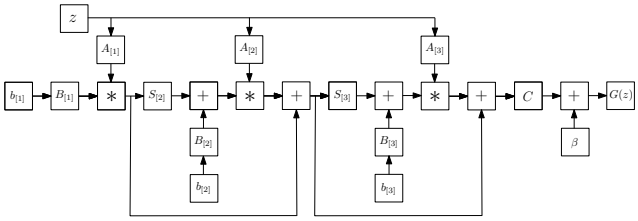

Figure 4: Schematic illustration of the NCP-Skip (for third order approximation). The difference from Fig. 3 is the skip connections added in this model.

3.2. Product of polynomials

Instead of using a single polynomial, we express the function approximation as a product of polynomials. The product is implemented as successive polynomials where the output of the i^{th} polynomial is used as the input for the $(i+1)^{th}$ polynomial. The concept is visually depicted in Fig. 5; each polynomial expresses a second order expansion. Stacking N such polynomials results in an overall order of 2^N. Trivially, if the approximation of each polynomial is B and we stack N such polynomials, the total order is B^N. The product does not necessarily demand the same order in each polynomial, the expressivity and the expansion order of each polynomial can be different and dependent on the task, e.g. for generative tasks that the resolution increases progressively, the expansion order could increase in the last polynomials. In all cases, the final order will be the product of each polynomial.

There are two main benefits of the product over the single polynomial: a) it allows using different decompositions (e.g. like in Sec. 3.1) and expressive power for each polynomial; b) it requires much less parameters for achieving the same

order of approximation. Given the benefits of the product of polynomials, we assume below that a product of polynomials is used, unless explicitly mentioned otherwise. The respective model of product polynomials is called ProdPoly.

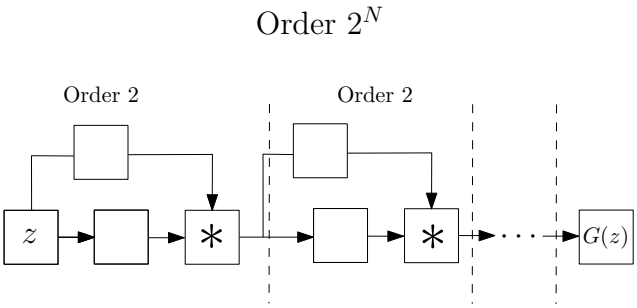

Figure 5: Abstract illustration of the ProdPoly. The input variable $\boldsymbol{z}$ on the left is the input to a 2^{nd} order expansion; the output of this is used as the input for the next polynomial (also with a 2^{nd} order expansion) and so on. If we use N such polynomials, the final output $G(\boldsymbol{z})$ expresses a 2^N order expansion. In addition to the high order of approximation, the benefit of using the product of polynomials is that the model is flexible, in the sense that each polynomial can be implemented as a different decomposition of Sec. 3.1.

3.3. Task-dependent input/output

The aforementioned polynomials are a function $\boldsymbol{x} = G(\boldsymbol{z})$, where the input/output are task-dependent. For a generative task, e.g. learning a decoder, the input $\boldsymbol{z}$ is typically some low-dimensional noise, while the output is a high-dimensional signal, e.g. an image. For a discriminative task the input $\boldsymbol{z}$ is an image; for a domain adaptation task the signal $\boldsymbol{z}$ denotes the source domain and $\boldsymbol{x}$ the target domain.

4. Proof of concept

In this Section, we conduct motivational experiments in both generative and discriminative tasks to demonstrate the expressivity of $\Pi-$nets. Specifically, the networks are implemented **without activation functions**, i.e. only linear operations (e.g. convolutions) and Hadamard products are used. In this setting, the output is linear or multi-linear with respect to the parameters.

4.1. Linear generation

One of the most popular generative models is Generative Adversarial Nets (GAN) [12]. We design a GAN, where the generator is implemented as a product of polynomials (using the NCP decomposition), while the discriminator of [37] is used. No activation functions are used in the generator, but a single hyperbolic tangent ($tanh$) in the image space [5].

[5] Additional details are deferred to the supplementary material.

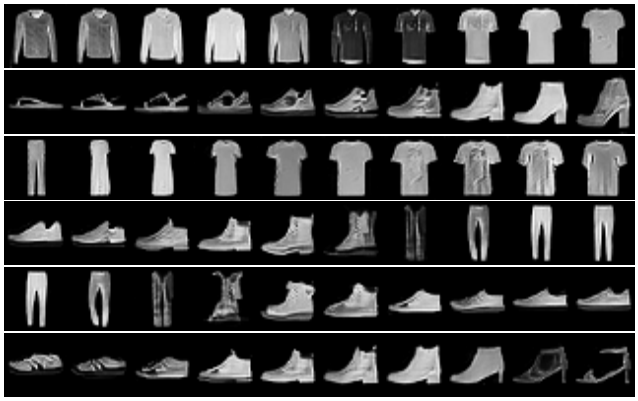

Figure 6: Linear interpolation in the latent space of ProdPoly (when trained on fashion images [64]). Note that the generator does not include any activation functions in between the linear blocks (Sec. 4.1). All the images are synthesized; the image on the leftmost column is the source, while the one in the rightmost is the target synthesized image.

Figure 7: Linear interpolation in the latent space of ProdPoly (when trained on facial images [10]). As in Fig. 6, the generator includes only linear blocks; the image on the leftmost column is the source, while the one in the rightmost is the target image.

Two experiments are conducted with a polynomial generator (Fashion-Mnist and YaleB). We perform a linear interpolation in the latent space when trained with Fashion-Mnist [64] and with YaleB [10] and visualize the results in Figs. 6, 7, respectively. Note that the linear interpolation generates plausible images and navigates among different categories, e.g. trousers to sneakers or trousers to t-shirts. Equivalently, it can linearly traverse the latent space from a fully illuminated to a partly dark face.

4.2. Linear classification

To empirically illustrate the power of the polynomial, we use ResNet without activations for classification. Residual Network (ResNet) [17, 54] and its variants [21, 62, 65, 69,

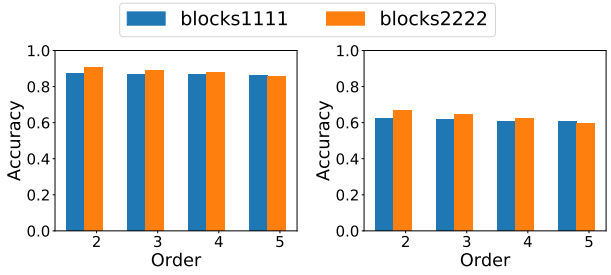

Figure 8: Image classification accuracy with linear residual blocks[6]. The schematic on the left is on CIFAR10 classification, while the one on the right is on CIFAR100 classification.

68] have been applied to diverse tasks including object detection and image generation [14, 15, 37]. The core component of ResNet is the residual block; the t^{th} residual block is expressed as $\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \boldsymbol{C}\boldsymbol{z}_t$ for input $\boldsymbol{z}_t$.

We modify each residual block to express a higher order interaction, which can be achieved with NCP-Skip. The output of each residual block is the input for the next residual block, which makes our ResNet a product of polynomials. We conduct a classification experiment with CIFAR10 [31] (10 classes) and CIFAR100 [30] (100 classes). Each residual block is modified in two ways: a) all the activation functions are removed, b) it is converted into an i^{th} order expansion with $i \in [2, 5]$. The second order expansion (for the t^{th} residual block) is expressed as $\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \boldsymbol{C}\boldsymbol{z}_t + \left(\boldsymbol{C}\boldsymbol{z}_t\right) * \boldsymbol{z}_t$; higher orders are constructed similarly by performing a Hadamard product of the last term with $\boldsymbol{z}_t$ (e.g., for third order expansion it would be $\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \boldsymbol{C}\boldsymbol{z}_t + \left(\boldsymbol{C}\boldsymbol{z}_t\right) * \boldsymbol{z}_t + \left(\boldsymbol{C}\boldsymbol{z}_t\right) * \boldsymbol{z}_t * \boldsymbol{z}_t$). The following two variations are evaluated: a) a single residual block is used in each 'group layer', b) two blocks are used per 'group layer'. The latter variation is equivalent to ResNet18 without activations.

Each experiment is conducted 10 times; the mean accuracy[5] is reported in Fig. 8. We note that the same trends emerge in both datasets[6]. The performance remains similar irrespective of the the amount of residual blocks in the group layer. The performance is affected by the order of the expansion, i.e. higher orders cause a decrease in the accuracy. Our conjecture is that this can be partially attributed to overfitting (note that a 3^{rd} order expansion for the 2222 block - in total 8 res. units - yields a polynomial of 3^8 power), however we defer a detailed study of this in a future version of our work. Nevertheless, in all cases without activations the accuracy is close to the original ResNet18 with activation functions.

[6] The performance of the baselines, i.e. ResNet18 **without** activation functions, is 0.391 and 0.168 for CIFAR10 and CIFAR100 respectively. However, we emphasize that the original ResNet was not designed to work without activation functions. The performance of ResNet18 in CIFAR10 and CIFAR100 **with** activation functions is 0.945 and 0.769 respectively.

Table 2: IS/FID scores on CIFAR10 [31] generation. The scores of [14, 15] are added from the respective papers as using similar residual based generators. The scores of [7, 19, 36] represent alternative generative models. ProdPolyoutperform the compared methods in both metrics.

Image generation on CIFAR10		
Model	IS (↑)	FID (↓)
SNGAN	8.06 ± 0.10	19.06 ± 0.50
NCP(Sec. 3.1)	8.30 ± 0.09	17.65 ± 0.76
ProdPoly	$\mathbf{8.49 \pm 0.11}$	$\mathbf{16.79 \pm 0.81}$
CSGAN-[14]	7.90 ± 0.09	-
WGAN-GP-[15]	7.86 ± 0.08	-
CQFG-[36]	8.10	18.60
EBM [7]	6.78	38.2
GLANN [19]	-	46.5 ± 0.20

5. Experiments

We conduct three experiments against state-of-the-art models in three diverse tasks: image generation, image classification, and graph representation learning. In each case, the baseline considered is converted into an instance of our family of Π-nets and the two models are compared.

5.1. Image generation

The robustness of ProdPoly in image generation is assessed in two different architectures/datasets below.

SNGAN on CIFAR10: In the first experiment, the architecture of SNGAN [37] is selected as a strong baseline on CIFAR10 [31]. The baseline includes 3 residual blocks in the generator and the discriminator.

The generator is converted into a Π-net, where each residual block is a single order of the polynomial. We implement two versions, one with a single polynomial (NCP) and one with product of polynomials (where each polynomial uses NCP). In our implementation $\boldsymbol{A}_{[n]}$ is a thin FC layer, $(\boldsymbol{B}_{[n]})^T\boldsymbol{b}_{[n]}$ is a bias vector and $\boldsymbol{S}_{[n]}$ is the transformation of the residual block. Other than the aforementioned modifications, the hyper-parameters (e.g. discriminator, learning rate, optimization details) are kept the same as in [37].

Each network has run for 10 times and the mean and variance are reported. The popular Inception Score (IS) [48] and the Frechet Inception Distance (FID) [18] are used for quantitative evaluation. Both scores extract feature representations from a pre-trained classifier (the Inception network [56]).

The quantitative results are summarized in Table 2. In addition to SNGAN and our two variations with polynomials, we have added the scores of [14, 15, 7, 19, 36] as reported in the respective papers. Note that the single polynomial already outperforms the baseline, while the ProdPoly boosts the performance further and achieves a substantial improvement over the original SNGAN.

Figure 9: Samples synthesized from ProdPoly (trained on FFHQ).

StyleGAN on FFHQ: StyleGAN [26] is the state-of-the-art architecture in image generation. The generator is composed of two parts, namely: (a) the mapping network, composed of 8 FC layers, and (b) the synthesis network, which is based on ProGAN [25] and progressively learns to synthesize high quality images. The sampled noise is transformed by the mapping network and the resulting vector is then used for the synthesis network. As discussed in the introduction StyleGAN is already an instance of the Π-net family, due to AdaIN. Specifically, the k^{th} AdaIN layer is $\boldsymbol{h}_k = (\boldsymbol{A}_k^T \boldsymbol{w}) * n(c(\boldsymbol{h}_{k-1}))$, where n is a normalization, c is a convolution and $\boldsymbol{w}$ is the transformed noise $\boldsymbol{w} = MLP(\boldsymbol{z})$ (mapping network). This is equivalent to our NCP model by setting $\boldsymbol{S}_{[k]}^T$ as the convolution operator.

In this experiment we illustrate how simple modifications, using our family of products of polynomials, further improve the representation power. We make a minimal modification in the mapping network, while fixing the rest of the hyperparameters. In particular, we convert the mapping network into a polynomial (specifically a NCP), which makes the generator a product of two polynomials.

The Flickr-Faces-HQ Dataset (FFHQ) dataset [26] which includes $70,000$ images of high-resolution faces is used. All the images are resized to 256×256. The best FID scores of the two methods (in 256×256 resolution) are $\mathbf{6.82}$ for ours and 7.15 for the original StyleGAN, respectively. That is, our method improves the results by 5%. Synthesized samples of our approach are visualized in Fig. 9.

5.2. Classification

We perform two experiments on classification: a) audio classification, b) image classification.

Audio classification: The goal of this experiment is twofold: a) to evaluate ResNet on a distribution that differs from that of natural images, b) to validate whether higher-order blocks make the model more expressive. The core assumption is that we can increase the expressivity of our model, or equivalently we can use less residual blocks of higher-order to achieve performance similar to the baseline.

The performance of ResNet is evaluated on the Speech Commands dataset [63]. The dataset includes $60,000$ audio files; each audio contains a single word of a duration of one second. There are 35 different words (classes) with each word having $1,500 - 4,100$ recordings. Every audio file is converted into a mel-spectrogram of resolution 32×32.

The baseline is a ResNet34 architecture; we use second-order residual blocks to build the Prodpoly-ResNet to match the performance of the baseline. The quantitative results are added in Table 3. The two models share the same accuracy, however Prodpoly-ResNet includes 38% fewer parameters. This result validates our assumption that our model is more expressive and with even fewer parameters, it can achieve the same performance.

Table 3: Speech classification with ResNet. The accuracy of the compared methods is similar, but Prodpoly-ResNet has 38% fewer parameters. The symbol '# par' abbreviates the number of parameters (in millions).

Speech Commands classification with ResNet			
Model	# blocks	# par	Accuracy
ResNet34	$[3, 4, 6, 3]$	21.3	0.951 ± 0.002
Prodpoly-ResNet	$[3, 3, 3, 2]$	**13.2**	0.951 ± 0.002

Image classification: We perform a large-scale classification experiment on ImageNet [47]. We choose float16 instead of float32 to achieve $3.5\times$ acceleration and reduce the GPU memory consumption by 50%. To stabilize the training, the second order of each residual block is normalized with a hyperbolic tangent unit. SGD with momentum 0.9, weight decay 10^{-4} and a mini-batch size of 1024 is used. The initial learning rate is set to 0.4 and decreased by a factor of 10 at $30, 60$, and 80 epochs. Models are trained for 90 epochs from scratch, using linear warm-up of the learning rate during first five epochs according to [13]. For other batch sizes due to the limitation of GPU memory, we linearly scale the learning rate (e.g. 0.1 for batch size 256).

The Top-1 error throughout the training is visualized in Fig. 10, while the validation results are added in Table 4. For a fair comparison, we report the results from our training in both the original ResNet and Prodpoly-ResNet[7]. Prodpoly-ResNet consistently improves the performance with an extremely small increase in computational complexity and model size. Remarkably, Prodpoly-ResNet50 achieves a single-crop Top-5 validation error of 6.358%, exceeding ResNet50 (6.838%) by 0.48%.

5.3. 3D Mesh representation learning

Below, we evaluate higher order correlations in graph related tasks. We experiment with 3D deformable meshes of fixed topology [45], *i.e.* the connectivity of the graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ remains the same and each different shape is defined as a different signal $\boldsymbol{x}$ on the vertices of the graph: $\boldsymbol{x} : \mathcal{V} \rightarrow \mathbb{R}^d$. As in the previous experiments, we extend a state-of-the-art operator, namely spiral convolutions [1], with the ProdPoly formulation and test our method on the

[7]The performance of the original ResNet [17] is inferior to the one reported here and in [20].

Table 4: Image classification (ImageNet) with ResNet. "Speed" refers to the inference speed (images/s) of each method.

ImageNet classification with ResNet					
Model	# Blocks	Top-1 error (%)	Top-5 error (%)	Speed	Model Size
ResNet50	$[3, 4, 6, 3]$	23.570	6.838	8.5K	50.26 MB
Prodpoly-ResNet50	$[3, 4, 6, 3]$	22.875	6.358	7.5K	68.81 MB

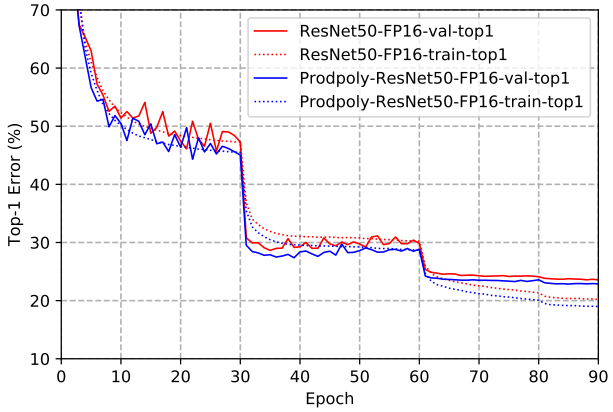

Figure 10: Top-1 error on ResNet50 and Prodpoly-ResNet50. Note that Prodpoly-ResNet performs consistently better during the training; the improvement is also reflected in the validation performance.

	error (mm) (↓)	speed (ms) (↓)
GAT [59]	0.732	11.04
FeastNet [60]	0.623	6.64
MoNet [38]	0.583	7.59
SpiralGNN [1]	0.635	4.27
ProdPoly (simple)	0.530	4.98
ProdPoly (simple - linear)	0.529	4.79
ProdPoly (full)	0.476	5.30
ProdPoly (full - linear)	**0.474**	5.14

Table 5: ProdPoly vs 1st order graph learnable operators for mesh autoencoding. Note that even without using activation functions the proposed methods significantly improve upon the state-of-the-art.

task of autoencoding 3D shapes. We use the existing architecture and hyper-parameters of [1], thus showing that ProdPoly can be used as a plug-and-play operator to existing models, turning the aforementioned one into a Spiral Π-Net. Our implementation uses a product of polynomials, where each polynomial is a specific instantiation of (4): $\boldsymbol{x}_n = \left(\boldsymbol{S}_{[n]}^T \boldsymbol{x}_{n-1}\right) * \left(\boldsymbol{S}_{[n]}^T \boldsymbol{x}_{n-1}\right) + \boldsymbol{S}_{[n]}^T \boldsymbol{x}_{n-1}, \boldsymbol{x} = \boldsymbol{x}_n + \boldsymbol{\beta}$, where $\boldsymbol{S}$ is the spiral convolution operator written in matrix form.[8] We use this model (*ProdPoly simple*) to showcase how to increase the expressivity without adding new blocks in the architecture. This model can be also re-interpreted as a learnable polynomial activation function as in [27]. We also show the results of our complete model (*ProdPoly full*), where $\boldsymbol{A}_{[n]}$ is a different spiral convolution.

In Table 5 we compare the reconstruction error of the autoencoder and the inference time of our method with the baseline spiral convolutions, as well as with the best results reported in [1] that other (more computationally involved - see inference time in table 5) graph learnable operators yielded. Interestingly, we manage to outperform all previously introduced models even when discarding the activation functions across the entire network. Thus, expressivity increases without having to increase the depth or the width of the architecture, as usually done by ML practitioners, and with small sacrifices in terms of inference time.

6. Discussion

In this work, we have introduced a new class of DCNNs, called Π-Nets that perform function approximation using a polynomial neural network. Our Π-Nets can be efficiently implemented via a special kind of skip connections that lead to high-order polynomials, naturally expressed with tensorial factors. The proposed formulation extends the standard compositional paradigm of overlaying linear operations with activation functions. We motivate our method by a sequence of experiments without activation functions that showcase the expressive power of polynomials, and demonstrate that Π-Nets are effective in both discriminative, as well as generative tasks. Trivially modifying state-of-the-art architectures in image generation, image and audio classification and mesh representation learning, the performance consistently imrpoves. In the future, we aim to explore the link between different decompositions and the resulting architectures and theoretically analyse their expressive power.

7. Acknowledgements

We are thankful to Nvidia for the hardware donation and Amazon web services for the cloud credits. The work of GC, SM, and GB was partially funded by an Imperial College DTA. The work of JD was partially funded by Imperial President's PhD Scholarship. The work of SZ was partially funded by the EPSRC Fellowship DEFORM: Large Scale Shape Analysis of Deformable Models of Humans (EP/S010203/1) and a Google Faculty Award. An early version with single polynomials for the generative settings can be found in [3].

[8] Stability of the optimization is ensured by applying vertex-wise instance normalization on the 2nd order term.

References

[1] Giorgos Bouritsas, Sergiy Bokhnyak, Stylianos Ploumpis, Michael Bronstein, and Stefanos Zafeiriou. Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *International Conference on Computer Vision (ICCV)*, 2019. 7, 8

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations (ICLR)*, 2019. 2

[3] Grigorios Chrysos, Stylianos Moschoglou, Yannis Panagakis, and Stefanos Zafeiriou. Polygan: High-order polynomial generators. *arXiv preprint arXiv:1908.06571*, 2019. 8

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 2

[5] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. In *Advances in neural information processing systems (NeurIPS)*, pages 2148–2156, 2013. 3

[6] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al. Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 395–408, 2017. 3

[7] Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models. In *Advances in neural information processing systems (NeurIPS)*, 2019. 6

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019. 3

[9] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 2

[10] Athinodoros S Georghiades, Peter N Belhumeur, and David J Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, (6):643–660, 2001. 5

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010. 2

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems (NeurIPS)*, 2014. 5

[13] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677*, 2017. 7

[14] Guillermo L Grinblat, Lucas C Uzal, and Pablo M Granitto. Class-splitting generative adversarial networks. *arXiv preprint arXiv:1709.07359*, 2017. 6

[15] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems (NeurIPS)*, pages 5767–5777, 2017. 2, 6

[16] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems (NeurIPS)*, pages 1135–1143, 2015. 3

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 2, 5, 7

[18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems (NeurIPS)*, pages 6626–6637, 2017. 6

[19] Yedid Hoshen, Ke Li, and Jitendra Malik. Non-adversarial image synthesis with generative latent nearest neighbors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5811–5819, 2019. 6

[20] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141, 2018. 7

[21] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 2, 6

[22] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *International Conference on Computer Vision (ICCV)*, pages 1501–1510, 2017. 2

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 2

[24] Alexey Grigorevich Ivakhnenko. Polynomial theory of complex systems. *transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971. 2

[25] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations (ICLR)*, 2018. 2, 7

[26] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2, 7

[27] Joe Kileel, Matthew Trager, and Joan Bruna. On the expressive power of deep polynomial neural networks. In *Advances in neural information processing systems (NeurIPS)*, 2019. 2, 8

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 2

[29] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009. 3

[30] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). 6

[31] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014. 6

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NeurIPS)*, pages 1097–1105, 2012. 1, 2

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 2

[34] Chien-Kuo Li. A sigma-pi-sigma neural network (spsnn). *Neural Processing Letters*, 17(1):1–19, 2003. 3

[35] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015. 2

[36] Thomas Lucas, Konstantin Shmelkov, Karteek Alahari, Cordelia Schmid, and Jakob Verbeek. Adversarial training of partially invertible variational autoencoders. *arXiv preprint arXiv:1901.01091*, 2019. 6

[37] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. 2, 5, 6

[38] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 8

[39] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 2

[40] S.M. Nikol'skii. *Analysis III: Spaces of Differentiable Functions*. Encyclopaedia of Mathematical Sciences. Springer Berlin Heidelberg, 2013. 3

[41] Sung-Kwun Oh, Witold Pedrycz, and Byoung-Jun Park. Polynomial neural networks architecture: analysis and design. *Computers & Electrical Engineering*, 29(6):703–725, 2003. 2

[42] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2337–2346, 2019. 2

[43] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Workshops*, 2017. 2

[44] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 2

[45] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3d faces using convolutional mesh autoencoders. In *European Conference on Computer Vision (ECCV)*, pages 704–720, 2018. 7

[46] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations (ICLR)*, 2018. 2

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 7

[48] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems (NeurIPS)*, pages 2234–2242, 2016. 6

[49] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2014. 2

[50] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015. 1, 2

[51] Yoan Shin and Joydeep Ghosh. The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *International Joint Conference on Neural Networks*, volume 1, pages 13–18, 1991. 3

[52] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017. 3

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 2

[54] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 1, 5

[55] Marshall H Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(5):237–254, 1948. 3

[56] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 6

[57] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *NeurIPS Workshops*, 2015. 2

[58] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 2

[59] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2018. 8

[60] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis.

In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 8

[61] Christodoulos Voutriaridis, Yiannis S Boutalis, and Basil G Mertzios. Ridge polynomial networks in pattern recognition. In *EURASIP Conference focused on Video/Image Processing and Multimedia Communications*, volume 2, pages 519–524, 2003. 3

[62] Wenhai Wang, Xiang Li, Jian Yang, and Tong Lu. Mixed link networks. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018. 6

[63] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018. 7

[64] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 5

[65] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 6

[66] Yan Xiong, Wei Wu, Xidai Kang, and Chao Zhang. Training pi-sigma network by online gradient algorithm with penalty for small weight update. *Neural computation*, 19(12):3356–3368, 2007. 3

[67] Chen Yunpeng, Jin Xiaojie, Kang Bingyi, Feng Jiashi, and Yan Shuicheng. Sharing residual units through collective tensor factorization in deep neural networks. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018. 3

[68] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 6

[69] Ke Zhang, Miao Sun, Tony X Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(6):1303–1314, 2017. 6

[70] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Learning hierarchical features from deep generative models. In *International Conference on Machine Learning (ICML)*, pages 4091–4099, 2017. 2