Training of Feedforward Networks Fails on a Simple Parity-Task

Sebastian Stabinger
Department of Computer Science
Universität Innsbruck
Technikerstrasse 21a, 6020 Innsbruck
`sebastian@stabinger.name`

David Peer
Department of Computer Science
Universität Innsbruck
Technikerstrasse 21a, 6020 Innsbruck
`d.peer@uibk.ac.at`

Antonio Rodríguez-Sánchez
Department of Computer Science
Universität Innsbruck
Technikerstrasse 21a, 6020 Innsbruck
`Antonio.Rodriguez-Sanchez@uibk.ac.at`

Abstract

Convolutional Neural Networks (CNNs) have become the gold standard for image classification over the past decade due to their robustness and high accuracy for many classification tasks. In this paper, we will present a new image classification dataset with only two classes, which we name the *parity dataset*, which can be easily solved by a CNN with handcrafted kernels. Despite this, preliminary experiments show that no solution can be found, if the weights of this architecture are initialized randomly, and the network is trained on data. In addition, commonly used CNN architectures like VGG, GoogLeNet, and ResNet completely fail at learning this task, even when given an unlimited number of training images. Based on the preliminary experiments and related work, we hypothesized that (1) state-of-the-art gradient descent methods on feedforward networks can not find a solution to the parity-task and (2) attention simplifies the parity-task.

1 Introduction

Convolutional Neural Networks (CNNs) have become the standard method for image classification over the last few years and achieve higher classification accuracy than humans on many datasets (e.g. as shown by He et al. [2015] on the ImageNet dataset). Despite their generally good performance, there are still datasets on which CNNs perform much worse than either older machine learning methods like Support Vector Machines or human test subjects. As Stabinger et al. [2016] could show, the SVRT dataset by Fleuret et al. [2011] is one of those datasets.

These findings inspired us to look for other datasets that are easily solved using simple algorithms, look unassuming to human observers, but are difficult to solve for currently used neural network architectures. One such dataset is the parity dataset we will present in this paper and that we want to analyze in the results paper in detail. Using the proposed dataset, we hope to highlight shortcomings in current neural network architectures and optimization methods in order to inspire research outside the trodden paths.

The paper is structured as follows: In the next section, related work is discussed. In Section 3.1 we introduce the parity dataset and in Section 3.2 we show that a perfect solution for this datasets exists if the weights of a proposed CNN architecture are set to specific values. We will present

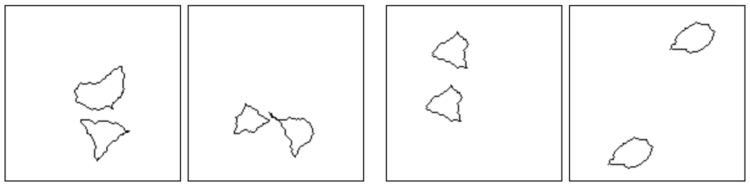

Figure 1: Examples of the two classes of problem 1 from the SVRT dataset by Fleuret et al. [2011]. In class 1, the two shapes are different, in class 2 they are identical.

preliminary experiments in Section 3.3 and show that a few already tested standard CNN architectures and optimizers cannot solve the parity dataset. Finally, in Section 4 we motivate and describe the experiments that we plan to do for the results paper.

2 Related work

The SVRT dataset by Fleuret et al. [2011] consists of abstract black and white images showing shape outlines. The images can be grouped into two classes, depending on whether the shapes fulfill some relational concept or not. The dataset consists of 23 problems with different relational concepts. As an example, images of problem 1 are of class 1 if they contain two shapes that are different and they are of class 2 if they contain two shapes that are identical. See Figure 1 for some example images of this problem.

Stabinger et al. [2016] could show that the tested CNNs (LeNet and GoogLeNet) are not able to achieve an accuracy above chance for problems for which the shapes have to be compared to each other. This finding was later confirmed by Ricci et al. [2018]. The original research by Fleuret et al. [2011], using handcrafted features and Support Vector Machines, as well as experiments with human subjects, did not show such a disparity in performance between problems that do need shape comparison and those that do not, indicating that CNNs seem to be especially ill-suited for this kind of tasks.

Later, Messina et al. [2019] as well as Funke et al. [2020] were able to solve the problems from the SVRT dataset using, among others, the ResNet architecture by He et al. [2016]. Still, the problems that require shape comparison need many more training images to be solved than the problems that do not. In addition, the PSVRT dataset by Kim et al. [2018], which replaces the shapes with patches that are randomly filled with black and white pixels, can only be solved with massive amounts of training data (> 300.000 training images), despite looking even simpler than the original problems from the SVRT dataset, indicating that the images of the SVRT dataset might not have the high variability as they seem at first sight. These findings inspired us to look for a seemingly easy dataset that is not solvable by CNNs even when using an unlimited number of training samples. We found such a dataset in the form of the parity dataset presented in this paper.

Using a very simple dataset to highlight shortcomings of current methods is not a novel idea. Liu et al. [2018] for example used a very simple dataset to highlight problems with the spacial invariance of convolutional layers and were able to propose solutions that also performed well for more commonly used datasets.

3 Methodology

3.1 The parity dataset

The proposed dataset, which we call the parity dataset, consists of gray scale images with a size of 14×14 pixels. On a background with a value of 0.5 (assuming 0 is black and 1 is white), three patches of size 3×3 pixels are randomly placed. The pixels of those patches are randomly colored in black or white (0 or 1 respectively). The *parity* of a patch is *odd* if it contains an odd number of black pixels and *even* if it contains an even number of black pixels. See Figure 2 for example images from this dataset.

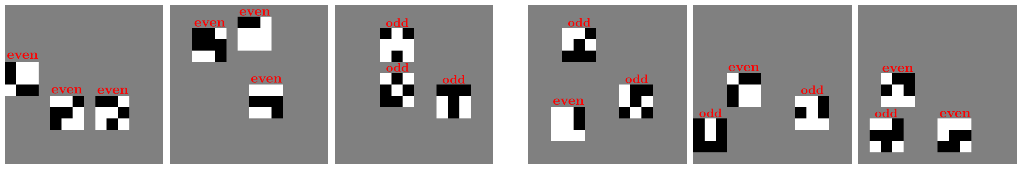

Figure 2: Example images of the proposed parity dataset. For class 1, the parity of all patches is the same, but for class 2, the parity of one patch is different from the parity of the rest of the patches. The parity of the patches are indicated in red above the patches for easier interpretability.

Images containing patches that all have the same parity (all odd or all even) are grouped in class 1 and images that contain patches with differing parity are grouped in class 2. The task is to learn this rule from labeled training images and to correctly classify previously unseen images.

3.2 Manually solving the dataset

To show that a solution exists, we will now present a manually constructed CNN that is able to solve the classification of the parity dataset.

This problem can be solved with four logical steps: (1) Count the number of white pixels for each patch. (2) One-hot encode this number across the depth dimension at each spatial patch position. (3) Multiply each odd depth dimension with two. (4) Output class 1 if the sum of all values is equal to three (all patches are even) or equal six (all patches are odd) and class 2 otherwise.

We call the depth dimension d and assume, without loss of generality, that background pixels are set to -100, each white pixel to 2 and each black pixel to 1. The afforementioned steps (1)-(3) are implemented with three convolutional layers and step (4) with three fully connected layers. To better visualize the output of each layer, we take a patch with 4 white pixels as an example, and will report the output vector $v \in \mathcal{R}^{12}$ of each layer at the spatial position centered on this patch:

Conv 1 The first convolutional filter with size 3×3 counts the white pixels. To achieve this, all weights of the filter are set to 1 which will result in the number of white pixels + 9 (each pixel adds one to the sum and two if it is white). Note: If the kernel is not exactly on the patch, the negative background value of -100 ensures that all values are negative such that after the ReLu operator the output is zero. This layer encodes each dimension so that it represents the distance to the value. Therefore, the bias for each filter d is set to $-(d+8)$. Input: 3×3 patch with 4 white pixels. Output after ReLu: $v_o = (5, 4, 3, 2, 1, 0, ..., 0)^T$.

Conv 2 The vector is now converted, so that all values > 0 are converted to 0s and all 0s are converted to 1s. To zero out all values larger than one we subtract the dimension $d-1$ from d. This explains why one additional depth dimension is needed. This can be implemented with a 1×1 kernel where each value is set to -1. To convert all 0s to 1s, we additionally add a bias term of one. Then the input $v_i = (5, 4, 3, 2, 1, 0, ..., 0)^T$ is converted into the output vector $v_o = (0, 0, 0, 0, 0, 1, ..., 1)^T$

Conv 3 To create a one-hot encoding at the correct position, we detect the edge between 0s and 1s. This can be implemented with a 1×1 filter by subtracting dimension $d+1$ from dimension d and multiplying the result with -1. This also explains why we need another additional depth dimension resulting in a total of 12 filters. Additionally, as mentioned previously, each odd dimension is multiplied with 2. Input: $v_i = (0, 0, 0, 0, 0, 1, ..., 1)^T$, Output: $v_o = (0, 0, 0, 0, 0, 1, 0, ..., 0)^T$. Note if the input patch contains e.g. five white pixels, then this output vector would be $v_o = (0, 0, 0, 0, 0, 0, 2, 0, ..., 0)^T$.

FC 1 This filter sums up all output values of the convolutional layer so that the output is ≤ 3 if all patches are even and ≥ 6 if all are odd. Otherwise, the value is $3 < x < 6$. To achieve this, no bias is needed and all weights are set to one.

FC 2 This layer indicates with output neuron 0 if the input was ≥ 6 and with output neuron 2 if the input was ≤ 3. Therefore, the weight to neuron 1 is set to 1 and the bias to -5 and the weight for neuron 2 to -1 and the bias to 4.

FC 3 The output of FC 2 is $(1, 0)$ if the patch contained only odd patches, $(0, 1)$ if all patches are even, and $(0, 0)$ otherwise. To linearly separate those cases, we set the output of neuron 1 constantly to 1. All weights of the second output neuron are set to 2 and the bias is set to zero. Therefore, if all patches are even or odd, the output of this layer is $(1, 2)$ and $(1, 0)$ otherwise and the problem is solved.

3.3 Preliminary experiments

As previously shown, the parity dataset can in principle be solved by a rather small CNN. Nonetheless, preliminary experiments show that a few tested standard CNN architectures seem to be completely unable to learn the underlying rule and at most, are only able to memorize examples from the training set.

For our preliminary experiments, we tested smaller, as well as commonly used bigger CNN architectures. The models were trained using the Lookahead optimizer by Zhang et al. [2019] with RAdam by Liu et al. [2019] as the base-optimizer (a combination which is generally known as Ranger, proposed by Wright [2019]). In addition, we also directly used RAdam as the optimizer. Tests were performed with six different learning rates, evenly spaced in the range from 10^{-1} to 10^{-6}.

For the smaller CNN architectures, the evaluation was performed on the original 14×14 sized images. The CNNs all start with one or multiple convolution blocks, consisting of a convolutional layer with a kernel size of 3×3 and $f(x) = \max(0, x)$ (ReLU) as the activation function, followed by a max pooling layer with a kernel size of 3×3. These convolutional blocks are followed by one or more fully connected layers. The number of convolutional blocks, number of kernels in the convolutional layers, number of fully connected layers and number of neurons per layer were varied to in order to evaluate CNNs of differing capacity. Considering that 512 different patches of size 3×3 exist, one of the tested architectures used 512 kernels in the first convolution layer. All weights were initialized using the method proposed by He et al. [2015] since it mitigates the chance of exploding or vanishing gradients when using ReLU activation functions. The handcrafted architecture proposed in Section 3.2, which is able to solve the parity dataset, was also tested on images with a size of 14×14 pixels.

As representatives of commonly used bigger CNNs, we trained the following architectures on the dataset: **VGG-16** by Simonyan and Zisserman [2014], **ResNet-18** and **ResNet-50** by He et al. [2016], and **GoogLeNet** by Szegedy et al. [2015]. Implementations of all architectures were taken from the model hub provided by the PyTorch framework by Paszke et al. [2019]. We used models that were already pre-trained on the ImageNet dataset by Krizhevsky et al. [2012] as well as randomly initialized models for all tested architectures. The pre-trained networks were used since previous experiments (e.g. on the SVRT and PSVRT dataset) showed that the networks converged faster even for images that look very different from the natural images that comprise ImageNet. Only the last layer of each architecture had to be changed to predict two instead of the 1000 classes from ImageNet.

We scaled up the images to the commonly expect input size of 224×224 pixels for these architectures, using nearest-neighbor interpolation. When upscaling the images, the task changes sligthly: We consider the image to be constructed from 14x14 squares (now each being 16x16 pixels big) instead of pixels and the goal is to detect whether a patch contains an even or odd number of squares instead of pixels.

All architectures were trained on 100.000 as well as an unlimited number of training images and tested on 10.000 images. We could observe two different outcomes during training: If the training set was small enough (100.000 images) and the capacity of the tested network was big enough, the training loss was converging towards 0, but the loss on the validation set was simultaneously increasing and the accuracy on the validation set did not significantly move above 0.5. This shows the architectures were only overfitting on the training set and were not able to generalize at all. The architectures were probably just memorizing the training data, which can sometimes be inferred when plotting the loss during training. In these cases, the loss increases slightly during one epoch and then sharply decreases when a new epoch starts and previously seen samples repeat again (see Figure 3).

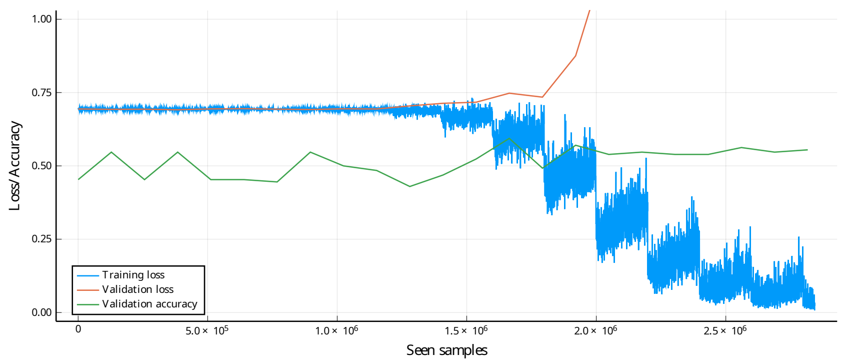

Figure 3: Example of the behaviour of the training and validation loss as well as the validation accuracy during a training run with a limited training set. The individual epochs can clearly be seen by the sharp decrease in the training loss. At this point, previously seen samples are repeated another time.

If we use an arbitrarily large training set by generating new samples on the fly, or if the capacity of the CNN is too small for the limited dataset with 100.000 images, the networks do not train at all. The training and validation loss, as well as the validation accuracy, stay constant during the whole training procedure, except for minor fluctuations.

4 Experimental protocol

4.1 Hypotheses

In the preliminary experiments, we were not able to find a single architecture that was able to learn anything useful regarding the task at hand, irrespective of the used optimizer, whether the network was pre-trained on ImageNet or not, or what learning rate was used. This even holds for the the architecture for which a handcrafted solution exists. If this architecture is initialized with random weights, it can also not be trained successfully on data either. Since we know that a solutions exist we hypothesize that:

Hypothesis 1 *Although a solution exist, state-of-the-art gradient descent methods, optimizing feed forward neural networks, cannot find a solution for the parity-task.*

To reduce the difficulty of the SVRT dataset for CNNs, Kim et al. [2018] used a pre-attention mechanism on the dataset by splitting the shapes into individual channels of the input. We therefore propose:

Hypothesis 2 *Distributing the three patches of an image to three input channels as a form of pre-attention, greatly simplifies the parity-task.*

In the next section we motivate experiments to test Hypothesis 1 and 2.

4.2 Hypothesis testing

To evaluate Hypothesis 1 we will train more standard architectures than in the preliminary experiments and we will also evaluate more optimizers (Ranger, Adam, and vanilla SGD) and vary the learning rate between 10^{-1} and 10^{-6}. Additionally, we will do a grid search for CNN architectures and we will include two orthogonal experiments. Namely, we will also test LSTMs and SVMs.

- *Experiment 1 - Standard architectures:* We will evaluate at least the following standard architectures for the dataset of size 224×224: VGG-16, VGG-19, ResNet-18, ResNet-34, ResNet-50, Inception-v4 from Szegedy et al. [2017] and EfficientNet from Tan and Le [2019]. Note that we will test each architecture with random initialization as well as a version pre-trained on ImageNet.

- *Experiment 2 - CNN grid search:* In Section 3.2 we have seen that the parity dataset with an image size of 14×14 can indeed be solved with classical CNN architectures. Therefore, we will do a grid search following this architecture [conv layers - fc layers - linear output] to test if optimizers are able to find this solution if hyperparameters are varied:
  - Number of CNN layers: $\{0, 1, 2, 3, 4\}$
  - Kernel sizes: $\{1 \times 1, 3 \times 3, 5 \times 5, 7 \times 7\}$
  - Number of kernels per layer $\{8, 16, 32, 64, 512\}$
  - Number of FC layers: $\{0, 1, 2, 3\}$
  - Number of neurons in the FC layers: $\{8, 64, 128, 512, 1024\}$
- *Experiment 3 - Orthogonal:* Our hypotheses are restricted to feedforward networks. Nonetheless, we are also interested to perform two orthogonal experiments. Namely, to evaluate the parity dataset with image sizes of 14×14 using Long Short Term Memory architectures as well as Support Vector Machines.

To evaluate Hypothesis 2 we must include a pre-attention mechanism. Therefore, we suggest the following experiment:

- *Experiment 4 - Pre-attention:* We will adapt the parity dataset so that each patch is presented in a different channel of the input. We will use this new, pre-attended parity dataset, to execute experiments 1-3 again.

In the next section, we want to present our expectations and the significance of possible outcomes.

4.3 Speculation on the results and their significance

Experiment 1 & 2 - CNN architectures Preliminary experiments have already shown that none of the tested standard CNNs can be trained on the parity task. As an outcome of experiment 1 and 2, we expect that none of the further tested architectures and optimizers will be able train on the parity dataset. Therefore, it would be highly interesting, should we be able to find an architecture or optimizer that is able to solve the problem. In this case, we would further investigate this special case in detail, which would guide future research into optimizers and/or architectures to solve this type of dataset. Otherwise, Hypothesis 1 will be further strengthened by the new experiments.

Experiment 3 - Orthogonal For the orthogonal experiments, our expectations are mixed: On one hand, we have seen that although a solution for CNNs exists, optimizers are not able to find a solution. Although there surely are solutions to the problem using LSTMs, the same restrictions might hold as for CNNs. On the other hand, there might be many more viable solutions that are possible with LSTMs, which might make it easier for an optimizer to reach one of those solutions. Both outcomes would be highly interesting and could guide future research. Experiments using SVMs could show that the parity task is not difficult to learn in principle, but is difficult to learn for neural networks using gradient descent.

Experiment 4 - Pre-attention As attentional mechanisms greatly reduce the difficulty of the SVRT dataset for CNNs as shown by Kim et al. [2018], we expect it might also help with the parity task. For the parity dataset, this case is interesting because we know that a solution exists (Section 3.2) and we could conclude that pre-attention either guides the optimizer to find a solution or increases the number of possible solutions. This can be further evaluated in future work depending on the outcome of this experiment.

References

François Fleuret, Ting Li, Charles Dubout, Emma K Wampler, Steven Yantis, and Donald Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108 (43):17621–17625, 2011.

Christina M Funke, Judy Borowski, Karolina Stosio, Wieland Brendel, Thomas SA Wallis, and Matthias Bethge. The notorious difficulty of comparing human and machine perception. *arXiv preprint arXiv:2004.09406*, 2020.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Dec 2015. doi: 10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Junkyung Kim, Matthew Ricci, and Thomas Serre. Not-so-clevr: learning same–different relations strains feedforward neural networks. *Interface Focus*, 8(4):20180011, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2019.

Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in neural information processing systems*, 31:9605–9616, 2018.

Nicola Messina, Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, and Claudio Gennaro. Testing deep neural networks on the same-different task. In *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, pages 1–6. IEEE, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Matthew Ricci, Junkyung Kim, and Thomas Serre. Same-different problems strain convolutional neural networks. *ArXiv180203390 Cs Q-Bio Available at: http://arxiv. org/abs/1802.03390 [Accessed May 28, 2018]*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sebastian Stabinger, Antonio Rodríguez-Sánchez, and Justus Piater. 25 years of cnns: Can we compare to human abstraction capabilities? *arXiv preprint arXiv:1607.08366*, 2016.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI '17, pages 4278–4284. AAAI Press, 2017.

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36nd International Conference on Machine Learning*, 2019.

Less Wright. New deep learning optimizer, ranger: Synergistic combination of radam + lookahead for the best of both. `https://bit.ly/2CS4OVJ`, August 2019.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pages 9597–9608, 2019.

Training of Feedforward Networks Fails on a Simple Parity-Task

Supplementary Material

Sebastian Stabinger, David Peer, and Antonio Rodríguez-Sánchez

November 27, 2020

1 Manual Implementation of a CNN that Solves the Dataset

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_addons as tfa

import numpy as np


class PerfectCNN(tf.keras.Model):


    def __init__(self, config):
        super(PerfectCNN, self).__init__()
        self.config = config
        self.convs = []
        self.fcs = []

        # INIT CONV1 layer
        kernel = np.ones(shape=[3,3,1,12])
        bias = np.ones(shape=[1,1,1,12])
        for i in range(12):
            bias[:,:,:,i] = -(i+8)

        self.convs.append(tf.keras.layers.Conv2D(
            filters=12,
            kernel_size=(3, 3),
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias),
            strides=1,
            padding="VALID",
            activation="relu"))

        # INIT CONV2 layer
        kernel = np.zeros(shape=[1,1,12,12])
        for i in range(12):
            kernel[:,:,i,i] = -1
        bias = np.ones(shape=[1,1,1,12])
        self.convs.append(tf.keras.layers.Conv2D(
            filters=12,
            kernel_size=(1, 1),
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias),
            strides=1,
            padding="VALID",
            activation="relu"))

        # INIT CONV3 layer
        kernel = np.zeros(shape=[1,1,12,12])
        for i in range(11):
            w = 2 if (i+1) % 2 == 0 else 1
            kernel[:,:,i,i] = -1 * w
            kernel[:,:,i+1,i] = 1 * w
        self.convs.append(tf.keras.layers.Conv2D(
            filters=12,
            kernel_size=(1, 1),
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.zeros_initializer(),
            strides=1,
            padding="VALID",
            activation="relu"))

        # Classify
        self.fcs.append(tf.keras.layers.Dense(
            units=1,
            activation="relu",
            use_bias=False,
            kernel_initializer=tf.ones_initializer()))

        kernel = [[1], [-1]]
        bias = [-5, 4]
        self.fcs.append(tf.keras.layers.Dense(
            units=2,
            activation="relu",
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias)))

        kernel = [[0, 2], [0, 2]]
        bias = [1, 0]
        self.fcs.append(tf.keras.layers.Dense(
            units=2,
            activation="linear",
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias)))


    def call(self, x, training=True):
        outputs = []
        batch_size = tf.shape(x)[0]

        # Conv layers
```

```
for conv in self.convs:
    x = conv(x)
    outputs.append(x)

# FC layers
x = tf.reshape(x, [batch_size, -1])
for fc in self.fcs:
    x = fc(x)
    outputs.append(x)

return outputs
```