

Assessment of CNN+XGBoost Performance for
Image Classification

ASSESSMENT OF CNN+XGBOOST PERFORMANCE FOR
IMAGE CLASSIFICATION

BY

ANDRII TURCHENKO, B.Eng.

A THESIS

SUBMITTED TO THE SCHOOL OF COMPUTATIONAL SCIENCE AND ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

© Copyright by Andrii Turchenko, January 2021

All Rights Reserved

Master of Science (2021)
(Computational Science & Engineering)

McMaster University
Hamilton, Ontario, Canada

TITLE: Assessment of CNN+XGBoost Performance for Image
Classification

AUTHOR: Andrii Turchenko
B.Eng., (Computer Engineering)
Lviv Polytechnic National University, Lviv, Ukraine

SUPERVISOR: Dr. Paul D. McNicholas

NUMBER OF PAGES: x, 73

To my family

Abstract

In the last few years, convolutional neural network (CNN) models have provided state-of-the-art results in visual recognition tasks. Similarly to CNNs, tree-based methods, in particular, gradient tree boosting (XGBoost) provided superior results in many applications. Taking into account the superiority of both methods, the goal of this work is to implement the CNN+XGBoost combined model where learned representations extracted from the CNN part will be used as input features for the XGBoost part. It is of particular interest to investigate whether the XGBoost part improves classification accuracy of the CNN part.

In this work, we use existing approaches — AlexNet, AllConvolutionalNet, WideResNet, DenseNet and CaffeNet (in transfer learning mode) — to extract features from the CNN part with different quality, which is defined by the classification accuracy of the appropriate CNN model. Then XGBoost is trained on the extracted features and the obtained final accuracy of AlexNet+XGBoost, AllConvolutionalNet+XGBoost, WideResNet+XGBoost, DenseNet+XGBoost and CaffeNet+XGBoost models are assessed. All experiments are fulfilled using the CIFAR10 image dataset. Our results show that features extracted by CNNs, which provided more than 85–88% classification accuracy, do not allow XGBoost to improve the final CNN+XGBoost classification performance.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Paul McNicholas of the Department of Mathematics and Statistics at McMaster University. His continuous advice, encouragement and support helped me a lot in going through the research process and allowed me to outperform many obstacles along the way.

Secondly, I would like to thank Dr. Fatima Batool from our research group for taking the time to review my work and her useful comments that allowed to improve it.

I would like to thank my old friends back in Ukraine and new friends here in Canada, who I had the pleasure of working with during my time as a Master's student.

Finally, I would like to thank my parents, grandparents, my brother and my girlfriend for their continuous love, support and encouragement throughout my years of study. This accomplishment would not have been possible without them.

Contents

Abstract	iv
Acknowledgements	v
1 Introduction	1
2 Related work	4
3 Background	7
3.1 Convolutional Neural Networks	7
3.2 XGBoost	17
3.3 Feature selection schemes	20
3.4 tSNE visualization algorithm	22
4 Experimental results	25
4.1 Hardware and software environment	25
4.1.1 Hardware and software	25
4.1.2 Caffe deep learning framework	26
4.2 CIFAR10 dataset	29
4.2.1 Original data	29

4.2.2	Augmented data	33
4.3	XGBoost results	35
4.4	AlexNet+XGBoost results	35
4.5	All Convolutional Net + XGBoost results	38
4.6	WideResNet + XGBoost results	39
4.7	DenseNet + XGBoost results on original data	41
4.8	DenseNet + XGBoost results on augmented data	43
4.9	Transfer Learning: CaffeNet pre-trained on ImageNet dataset + XG- Boost results	45
4.10	Analysis and summary of results	46
5	Conclusions	55
A	Confusion matrices for all classification results	58
	Bibliography	66

List of Tables

2.1	Accuracy comparison of CNN and NBDT approach (Wan et al., 2020).	6
4.1	Output HDF5 structures of CIFAR10 training and testing datasets.	32
4.2	AlexNex summary.	35
4.3	XGBoost accuracy using 256 <submean> AlexNet features.	37
4.4	AllConvNet summary.	38
4.5	XGBoost accuracy using 64 <original> AllConvNet features.	39
4.6	WideResNet summary.	39
4.7	XGBoost accuracy using 640 <original> WideResNet features.	40
4.8	DenseNet summary.	41
4.9	XGBoost accuracy using 448 <submean> DenseNet features.	42
4.10	XGBoost accuracy using all feature selection schemes for DenseNet features obtained in 5-fold CV fashion.	43
4.11	XGBoost accuracy using 448 <submean> features extracted by DenseNet on augmented dataset in one-module and 5-fold CV fashions.	44
4.12	CaffeNet summary.	46
4.13	XGBoost accuracy using 200 <submean> <RRC> CaffeNet features.	46
4.14	tSNE visualizations of extracted features from CNN parts.	51
4.15	Summary of classification results.	54

List of Figures

3.1	Graphical representation of convolutional layer.	8
3.2	Comparison between ReLU (a) and Leaky ReLU (b).	10
3.3	Graphical representation of max and average pooling.	11
4.1	Descriptive examples of layers in Caffe.	28
4.2	Example of the solver file in Caffe.	30
4.3	Random visual check of CIFAR10 classes while creating HDF5 training and testing datasets.	33
4.4	Example of data augmentation.	34
4.5	Graphical representation of AlexNet CNN.	36
4.6	DenseNet vs XGBoost classification accuracy at different feature ex- traction quality.	53
4.7	XGBoost training time at different feature extraction quality.	53
A.1	Confusion matrices for AlexNet+XGBoost experiment.	59
A.2	Confusion matrices for AllConvNet + XGBoost experiment.	59
A.3	Confusion matrices for WideResNet+XGBoost experiment.	60
A.4	Confusion matrices for DenseNet+XGBoost one-module experiment on original data.	61

A.5	Confusion matrices for DenseNet+XGBoost 5-fold CV experiment on original data.	62
A.6	Confusion matrices for DenseNet+XGBoost one-module experiment on the augmented data.	63
A.7	Confusion matrices for DenseNet+XGBoost 5-fold CV experiment on the augmented data.	64
A.8	Confusion matrices for CaffeNet+XGBoost transfer learning one-module experiment.	65

Chapter 1

Introduction

In the last few years, deep learning has led to very good performance on a variety of problems, such as visual recognition, speech recognition and natural language processing. Among different types of deep neural networks, convolutional neural networks (CNNs) have provided state-of-the-art results in visual recognition tasks and, therefore, they have been extensively studied (e.g., Goodfellow et al., 2016; LeCun et al., 2015, 1998; Krizhevsky et al., 2012; Springenberg et al., 2014; Zagoruyko and Komodakis, 2016; Huang et al., 2016; Gu et al., 2018). In the field of computer vision, CNNs were the winning architecture in all ImageNet Large Scale Visual Recognition Competition (LSVRC) challenges from 2010 to 2017¹.

Similarly to CNNs, gradient tree boosting (Friedman, 2001), is another technique that shines in many applications (Chen and Guestrin, 2016). Tree boosting has been shown to give state-of-the-art results on many standard classification benchmarks (Robust, 2010). LambdaMART (Burges, 2010) achieves state-of-the-art result for

¹<http://www.image-net.org/challenges/LSVRC/>, accessed November 2020.

ranking problems. A tree boosting technique was incorporated into real-world production pipelines for ad click-through rate prediction (He et al., 2014). It is the de-facto choice of ensemble method and has been successful in challenges such as the Netflix prize (Bennett and Lanning, 2007). A very popular implementation of tree boosting technique called XGBoost — extreme gradient boosting — is available as an open source package². The impact of the tree boosting techniques in general, and XGBoost package in particular, has been widely recognized in a number of machine learning and data mining challenges. For example, among the 29 challenge winning solutions published at Kaggle’s blog³ during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural networks in ensembles. The success of XGBoost was also witnessed in KDDCup 2015, where XGBoost was used by every winning team in the top-10. Moreover, the winning teams reported that ensemble methods outperform a well-configured XGBoost by only a small amount (Bekkerman, 2015).

Taking into account the excellence of both aforementioned approaches, CNN and XGBoost, the goal of this work is to implement a CNN+XGBoost combined model where learned representations extracted from the CNN part will be used as input features for the XGBoost part. In particular, we investigate whether the XGBoost part allows improved original classification accuracy of the CNN part. We choose the CIFAR10 image dataset to be used in our experimental research.

This thesis is organized as follows. In Chapter 2, we introduce related work. Chapter 3 overviews theoretical approaches behind all methods used in this work and

²<https://xgboost.readthedocs.io/en/latest/index.html>, accessed November 2020.

³<https://www.kaggle.com/>, accessed November 2020.

consists of four paragraphs. Section 3.1 presents the basic principles of CNNs. Section 3.2 considers the ADABOOST algorithm as an example of a gradient tree boosting method. Section 3.3 provides details of used filtering feature selection schemes based on (i) redundancy removed correlated (<RRC>) features, (ii) Random Forest ranks (<RFRANKS>), and (iii) minimum Redundancy Maximum Relevance (<mRMR>) method. Section 3.4 describes t-Distributed Stochastic Neighbor Embedding (t-SNE) as an unsupervised, non-linear technique used for visualizing high-dimensional data. Chapter 4 presents experimental results. Section 4.1 shows the hardware and software environments used in this work including details of the Caffe deep learning framework. Section 4.2 provides details of creation of working original and augmented versions of CIFAR10 image dataset. Sections 4.3–4.9 present experimental results for different CNN architectures used as the CNN part in the CNN+XGBoost combination. Section 4.10 provides analysis and summary of obtained experimental results. Finally, Chapter 5 concludes the thesis and suggests future research directions.

Chapter 2

Related work

A summary of a performance of different methods on the CIFAR10 dataset is presented in Benenson’s website¹. Because, methodically, CNNs provide better performance on images, the majority of those results came from (i) supervised deep CNNs which use images directly, (ii) useful features extracted from images by deep auto-encoders in unsupervised way, and (iii) other techniques which also involve CNNs in some ways. Classification accuracies given by Benenson¹ vary from the 96.53% in the best case (Graham, 2015) to 75.86% in the worst case (McDonnell and Vladusich, 2015).

The classification accuracy provided by XGBoost model in generally could be much worse because XGBoost does not treat imaging information as spatial images, when location of pixels plays a role and determines an image itself, but as a set of “independent” features which are the values of appropriate pixels. For example, a CIFAR10 image has 32×32 pixels and 3 RGB channels, and could be presented

¹https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130, accessed November 2020.

to XGBoost as one dimensional vector of 3,072 elements. Ponomareva et al. (2017) provided two improvements to the standard tree boosting algorithm by (i) changing the boosting formalism from scalar-valued trees to vector-valued trees and (ii) introducing layer-by-layer boosting which leads to faster convergence and a more compact ensemble. The authors implemented their improvements in the open-source TensorFlow Boosted Trees (TFBT) package and demonstrated the efficacy on a variety of multiclass datasets. Their classification accuracies on the CIFAR10 dataset are: 40.50% by default; 41.94% by tuned XGBoost models from the standard Scikit-Learn Python library; and 48.96% by their TFBT package when they used 100 trees and `depth=4` in each of experiments.

Ren et al. (2017) presented a combined deep CNN+XGBoost model, where a deep CNN has performed feature extraction from images and XGBoost has performed classification taking the extracted features from the fully-connected layer of the deep CNN. The authors showed an improvement in classification accuracy from 76.28% (for a standard deep CNN model) to 80.77% (for the combined deep CNN+XGBoost model) for the CIFAR10 dataset. However, the authors did not specify exactly from which of fully-connected layers of deep CNN they took the extracted features because a classic deep CNN model for image recognition has at least two last fully-connected layers (Krizhevsky et al., 2012).

Wan et al. (2020) presented neural-backed decision trees (NBDT) approach. They use pre-trained CNN models to get features for a tree-based model. The authors showed that their NBDT method can achieve accuracy: (i) within 1% of the base CNN model on CIFAR10, CIFAR100 and TinyImageNet datasets using recently presented state-of-the-art WideResNet CNN; and (ii) within 2% if using pre-trained EfficientNet

on ImageNet dataset. However, the performance of the NBDT approach was slightly deteriorated (see Table 2.1).

Table 2.1: Accuracy comparison of CNN and NBDT approach (Wan et al., 2020).

Method	Backbone	CIFAR10	CIFAR100
CNN	WideResNet 28x10	97.62%	82.09%
NBDT	WideResNet 28x10	97.57%	82.87%
CNN	ResNet18	94.97%	75.92%
NBDT	ResNet18	94.67%	74.92%

Chapter 3

Background

3.1 Convolutional Neural Networks

There are numerous variants of CNN architectures in the literature (e.g., Goodfellow et al., 2016; LeCun et al., 2015, 1998; Krizhevsky et al., 2012; Springenberg et al., 2014; Zagoruyko and Komodakis, 2016; Huang et al., 2016; Gu et al., 2018). However, their basic components are very similar (Gu et al., 2018). Taking the famous LeNet-5 as an example, it consists of three types of layers, namely: convolutional, pooling, and fully-connected layers (LeCun et al., 1998). The convolutional layer is composed of several convolution kernels which are used to compute different feature maps. Specifically, each neuron of a feature map is connected to a region of neighbouring neurons in the previous layer. Such a neighbourhood is referred to as the neuron's receptive field (input patch) in the previous layer. The new feature map can be obtained by first convolving the input with a learned kernel and then applying an element-wise nonlinear activation function on the convolved results. To generate each feature map, the kernel is shared by all spatial locations of the input. The complete feature maps

are obtained by using several different kernels.

Mathematically, the feature value at location (i, j) in the k th feature map of l th layer, $z_{i,j,k}^l$, is calculated by

$$z_{i,j,k}^l = \mathbf{w}_k^{lT} \mathbf{x}_{i,j}^{l-1} + b_k^l,$$

where w_k^l and b_k^l are the weight vector and bias term of the k th filter of the l th layer respectively, and $x_{i,j}^{l-1}$ is the input patch centred at location (i, j) of the l th-1 layer (see Figure 3.1). Note that the kernel w_k^l that generates the feature map $z_{:, :, k}^l$ is shared. Such a weight sharing mechanism has several advantages such as it can reduce the model complexity and make the network easier to train.

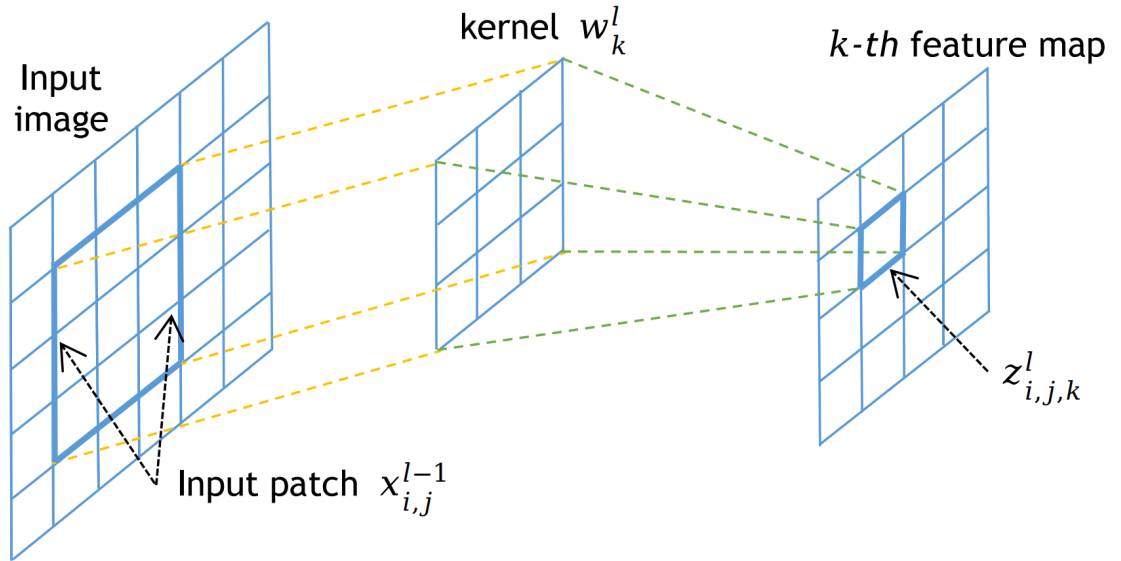


Figure 3.1: Graphical representation of convolutional layer.

The activation function introduces nonlinearities to CNN, which are desirable for multi-layer networks to detect nonlinear features. Let $a(\cdot)$ denote the nonlinear activation function. The activation value $a_{i,j,k}^l$ of convolutional feature $z_{i,j,k}^l$ can be

computed as

$$a_{i,j,k}^l = a(z_{i,j,k}^l).$$

Typical activation functions are Sigmoid, Tanh (LeCun et al., 2012) and Rectified linear unit (ReLU) (Nair and Hinton, 2010). ReLU is one of the most notable non-saturated activation functions. The ReLU activation function is defined as

$$a_{i,j,k} = \max(z_{i,j,k}, 0),$$

where $z_{i,j,k}$ is the input of the activation function at location (i, j) on the k th channel. ReLU is a piecewise linear function which prunes the negative part to zero and retains the positive part (see Figure 3.2a). The simple $\max(\cdot)$ operation of ReLU allows it to compute much faster than Sigmoid or Tanh activation functions, and it also induces the sparsity in the hidden units and allows the network to easily obtain sparse representations. It has been shown that deep networks can be trained efficiently using ReLU even without pre-training (Russakovsky et al., 2015). Even though the discontinuity of ReLU at 0 may hurt the performance of back-propagation, many works have shown that ReLU works better than Sigmoid and Tanh activation functions empirically (Maas et al., 2013; Zeiler et al., 2013).

A potential disadvantage of ReLU unit is that it has zero gradient whenever the unit is not active. This may cause units that do not active initially to never active as the gradient-based optimization will not adjust their weights. Also, it may slow down the training process due to the constant zero gradients. To alleviate this problem,

Maas et al. (2013) introduced Leaky ReLU which is defined as

$$a_{i,j,k} = \max(z_{i,j,k}, 0) + \lambda \min(z_{i,j,k}, 0),$$

where $\lambda \in (0, 1)$ is a predefined parameter. Compared with ReLU, Leaky ReLU compresses the negative part rather than mapping it to constant zero, which makes it allow for a small, non-zero gradient when the unit is not active (see Figure 3.2b).

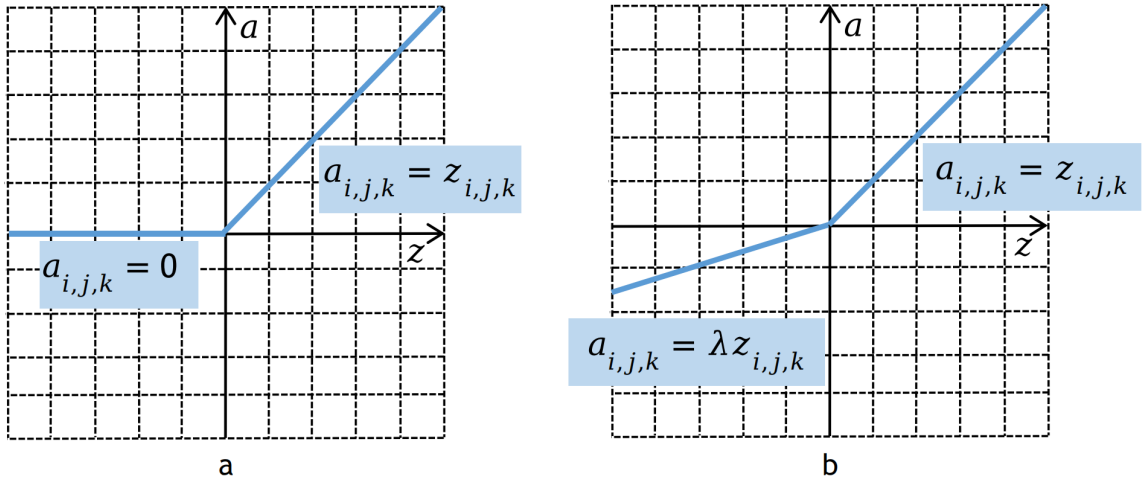


Figure 3.2: Comparison between ReLU (a) and Leaky ReLU (b).

The pooling layer aims to achieve shift-invariance by reducing the resolution of the feature maps. It is usually placed between two convolutional layers. Each feature map of a pooling layer is connected to its corresponding feature map of the preceding convolutional layer.

Denoting the pooling function as $\text{pool}(a_{:, :, k}^l)$, for each feature map $a_{:, :, k}^l$, we have

$$y_{i,j,k}^l = \text{pool}(a_{m,n,k}^l), \forall (m, n) \in \mathcal{R}_{ij},$$

where \mathcal{R}_{ij} is a local neighbourhood around location (i, j) . The typical pooling operations are max pooling (Boureau et al., 2010) and average pooling (Wang et al., 2012). Pooling layer makes the features robust against noise and distortion. Figure 3.3 shows an example, how a pooling layer fulfills the pooling function: the input has a size 4×4 elements, for 2×2 pooling, 4×4 image is divided into four non-overlapping matrices of size 2×2 (filled by different colors). In the case of max pooling, the output is the maximum value of the four values in each 2×2 matrix. In case of average pooling, the output is the average of the four values in each 2×2 matrix (Hijazi et al., 2015).

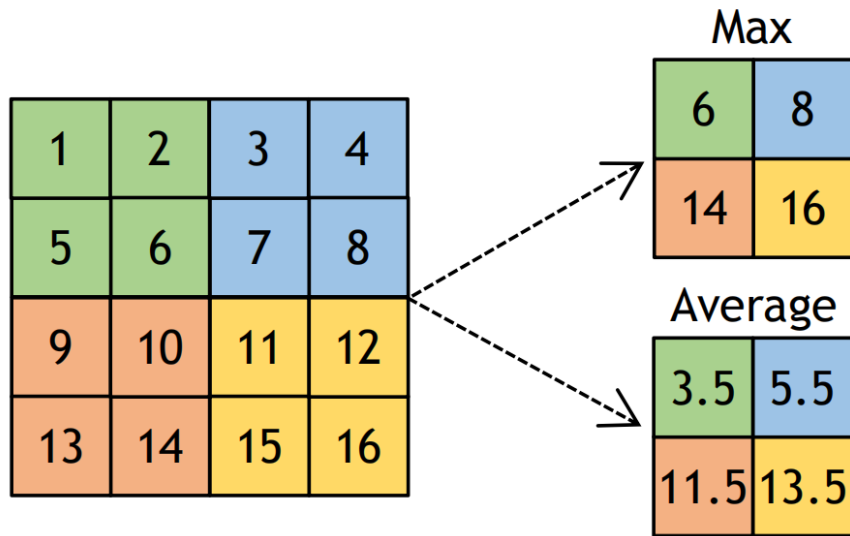


Figure 3.3: Graphical representation of max and average pooling.

The kernels in the 1st convolutional layer of a CNN are designed to detect low-level features such as edges and curves, while the kernels in higher layers are learned to encode more abstract features. By stacking several convolutional and pooling layers, we could gradually extract higher-level feature representations. After several convolutional and pooling layers, there may be one or more fully-connected layers

which aim to perform high-level reasoning (Simonyan and Zisserman, 2015; Zeiler and Fergus, 2014; Hinton et al., 2012). They take all neurons in the previous layer and connect them to every single neuron of current layer to generate global semantic information. Also, a fully-connected layer is not always necessary as it can be replaced by a 1×1 convolution layer (Lin et al., 2014). The last layer of CNNs is an output layer. For classification tasks, the softmax loss is commonly used (Russakovsky et al., 2015). It is essentially a combination of multinomial logistic loss and softmax. Given a training set $\{(x^{(i)}, y^{(i)}) ; i \in 1, \dots, N, y^{(i)} \in 1, \dots, K\}$, where $x^{(i)}$ is the i th input image patch, and $y^{(i)}$ is its target class label among the K classes. The prediction of j th class for i th input is transformed with the softmax function:

$$p_j^{(i)} = \frac{e^{z_j^{(i)}}}{\sum_{l=1}^K e^{z_l^{(i)}}},$$

where $z_j^{(i)}$ is usually the activations of a densely connected layer, so $z_j^{(i)}$ can be written as $z_j^{(i)} = w_j^T a^{(i)} + b_j$. Softmax turns the predictions into non-negative values and normalizes them to get a probability distribution over classes. Such probabilistic predictions are used to compute the multinomial logistic loss, i.e., the softmax loss, as follows:

$$\mathcal{L}_{\text{softmax}} = -\frac{1}{N} \left[\sum_{i=1}^N \sum_{j=1}^K 1 \{ \mathbf{y}^{(i)} = j \} \log p_j^{(i)} \right].$$

Training a CNN is a problem of global optimization. By minimizing the loss function, we can find the best fitting set of parameters. Stochastic gradient descent (SGD) is a common solution for optimizing CNNs (Wijnhoven and de With, 2010; Zinkevich et al., 2010). The back-propagation algorithm is the standard training method that uses gradient descent to update the parameters. Many gradient descent

optimization algorithms have been proposed (Qian, 1999; Kingma and Ba, 2015). Standard gradient descent algorithm update the parameters θ of the objective $L(\theta)$ as

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} E[L(\theta)],$$

where $E[L(\theta)]$ is the expectation of $L(\theta)$ over the full training set and η is the learning rate. Instead of computing $E[L(\theta)]$, SGD estimates the gradients on the basis of a single randomly picked example $(x^{(t)}, y^{(t)})$ from the training set

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t; \mathbf{x}^{(t)}, \mathbf{y}^{(t)})$$

(Wijnhoven and de With, 2010).

In practice, each parameter update in SGD is computed with respect to a mini-batch as opposed to a single example. This could help to reduce the variance in the parameter update and can lead to more stable convergence. The convergence speed is controlled by the learning rate η_t . However, mini-batch SGD does not guarantee good convergence, and there are still some challenges that need to be addressed. Firstly, it is not easy to choose a proper learning rate. One common method is to use a constant learning rate that gives stable convergence in the initial stage, and then reduce the learning rate as the convergence slows down. Additionally, learning rate schedules have been proposed to adjust the learning rate during the training (Loshchilov and Hutter, 2017; Schaul et al., 2013). To make the current gradient update depend on historical batches and accelerate training, Qian (1999) proposed momentum to accumulate a velocity vector in the relevant direction. The classical

momentum update is given by

$$\begin{aligned}\mathbf{v}_{t+1} &= \gamma \mathbf{v}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t; \mathbf{x}^{(t)}, \mathbf{y}^{(t)}), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \mathbf{v}_{t+1},\end{aligned}$$

where \mathbf{v}_{t+1} is the current velocity vector and γ is the momentum term which is usually set to 0.9. Nesterov momentum (Sutskever et al., 2013) is another way of using momentum in gradient descent optimization:

$$\mathbf{v}_{t+1} = \gamma \mathbf{v}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L} | (\boldsymbol{\theta}_t + \gamma \mathbf{v}_t; \mathbf{x}^{(t)}, \mathbf{y}^{(t)}).$$

Compared with the classical momentum (Qian, 1999), which first computes the current gradient and then moves in the direction of the updated accumulated gradient, Nesterov momentum first moves in the direction of the previous accumulated gradient $\gamma \mathbf{v}_t$, calculates the gradient and then makes a gradient update. This anticipatory update prevents the optimization from moving too fast and achieves better performance (Zhang et al., 2015).

Over-fitting is a non-negligible problem in deep CNNs, which can be effectively reduced by regularization. Several effective regularization techniques are available and include l_p -norm, Dropout, and DropConnect. The l_p -norm regularization technique modifies the objective function by adding additional terms that penalize the model complexity. Formally, if the loss function is $L(\theta, x, y)$, then the regularized loss will be

$$E(\theta, \mathbf{x}, \mathbf{y}) = \mathcal{L}(\theta, \mathbf{x}, \mathbf{y}) + \lambda R(\theta),$$

where $R(\theta)$ is the regularization term and λ is the regularization strength. The l_p -norm regularization function is usually employed as

$$R(\theta) = \sum_j \|\theta_j\|_p^p.$$

When $p \geq 1$, the l_p -norm is convex, which makes the optimization easier and renders this function attractive (Hinton et al., 2012). For $p = 2$, the l_2 -norm regularization is commonly referred to as weight decay. Hinton et al. (2012) also apply Dropout to fully-connected layers. The output of Dropout is

$$\mathbf{y} = \mathbf{r} * a(\mathbf{W}^T \mathbf{x}),$$

where $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ is the input to fully-connected layer, $\mathbf{W} \in \mathbb{R}^{n \times d}$ is a weight matrix, and \mathbf{r} is a binary vector of size d whose elements are independently drawn from a Bernoulli distribution with parameter p , i.e., $r_i \sim \text{Bernoulli}(p)$ for $i = 1, \dots, d$, and "*" denotes element-wise product. Dropout can prevent the network from becoming too dependent on any one (or any small combination of) neurons, and can force the network to be accurate even in the absence of certain information.

Deep CNNs are particularly dependent on the availability of large quantities of training data. An elegant solution to alleviate the relative scarcity of the data compared to the number of parameters involved in CNNs is data augmentation (Rusakovsky et al., 2015). Data augmentation consists in transforming the available data into new data without altering their natures. Popular augmentation methods include simple geometric transformations such as sampling, mirroring, rotating, shifting, and various photometric transformations.

Deep CNNs have a huge amount of parameters and their loss functions are non-convex (Choromanska et al., 2015), which makes them very difficult to train. To achieve a fast convergence in training and avoid the vanishing gradient problem, a proper network initialization is one of the most important prerequisites (Mishkin and Matas, 2016; Sutskever et al., 2013). The bias parameters can be initialized to zero, while the weight parameters should be initialized carefully to break the symmetry among hidden units of the same layer. If the network is not properly initialized, e.g., each layer scales its input by k , the final output will scale the original input by k^L , where L is the number of layers. In this case, the value of $k > 1$ leads to extremely large values of output layers while the value of $k < 1$ leads a diminishing output value and gradients. Krizhevsky et al. (2012) initialize the weights of their network from a zero-mean Gaussian distribution with standard deviation 0.01 and set the bias terms of the second, fourth and fifth convolutional layers as well as all the fully-connected layers to constant one. Another famous random initialization method is “Xavier”, which is proposed by Glorot and Bengio (2010). They pick the weights from a Gaussian distribution with mean 0 and a variance of

$$\frac{2}{n_{\text{in}} + n_{\text{out}}},$$

where n_{in} is the number of neurons feeding into it and n_{out} is the number of neurons the result is fed to. Thus “Xavier” can automatically determine the scale of initialization based on the number of input and output neurons, and keep the signal in a reasonable range of values through many layers. One of its variants in Caffe uses the n_{in} -only variant, which makes it easier to implement.

Data normalization is usually the first step of data preprocessing. Global data

normalization transforms all the data to have mean 0 and variance 1. However, as the data flows through a deep network, the distribution of input to internal layers will be changed, which will lose the learning capacity and accuracy of the network. Ioffe and Szegedy (2015) propose an efficient method called batch normalization (BN) to partially alleviate this phenomenon. It accomplishes the so-called covariate shift problem by a normalization step that fixes the means and variances of layer inputs where the estimations of mean and variance are computed after each mini-batch rather than the entire training set.

3.2 XGBoost

XGBoost is derived from a boosting method, which is one of the most powerful learning ideas introduced in the last twenty years (Hastie et al., 2008). The motivation for boosting was a procedure that combines the outputs of many “weak” classifiers to produce a powerful “committee”. From this perspective boosting bears a resemblance to bagging and other committee-based approaches; however, boosting is fundamentally different (Hastie et al., 2008; McNicholas and Tait, 2019). Let us consider XGBoost on the example of the most popular boosting algorithm called “AdaBoost.M1” (Freund and Schapire, 1997). Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictor variables X , a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^N I(y_i \neq G(x_i)) \quad (3.1)$$

and the expected error rate on future predictions is $E_{XY}I(Y \neq G(X))$.

A weak classifier is one whose error rate is only slightly better than random guessing. The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x), m = 1, \dots, M$.

The predictions from all of weak classifiers are then combined through a weighted majority vote to produce the final prediction

$$G(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m G_m(x) \right),$$

where $\alpha_1, \dots, \alpha_M$ are computed by the boosting algorithm, and weight the contribution of each respective $G_m(x)$. Their effect is to give higher influence to the more accurate classifiers in the sequence.

Algorithm 1 shows a pseudo-code of the AdaBoost algorithm. The data modifications at each boosting step consist of applying weights w_1, \dots, w_N to each of the training observations $(x_i, y_i), i = 1, \dots, N$. Initially all of the weights are set to $w_i = 1/N$, so that the first step simply trains the classifier on the data in the usual manner. For each successive iteration $m = 2, \dots, M$, the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. At step m , those observations that were mis-classified by the classifier $G_{m-1}(x)$ induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly. Thus as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. Each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence.

The success of boosting idea lies in expression (3.1). Boosting is a way of fitting

1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \dots, N$.
 2. For $m = 1$ to M :
 - (a) Fit a classifier $G_m(x)$ to the training data using weights w_i .
 - (b) Compute

$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
 - (c) Compute $\alpha_m = \log((1 - \text{err}_m) / \text{err}_m)$.
 - (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \dots, N$.
 3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.
-

Algorithm 1: The AdaBoost.M1 algorithm.

an additive expansion in a set of elementary “basis” functions. Here the basis functions are the individual models of classifiers $G_m(x) \in \{-1, 1\}$. More generally, basis function expansions take the form

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m),$$

where $\beta_m, m = 1, \dots, M$ are the expansion coefficients and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x , characterized by a set of parameters γ . Typically these individual models are fit by minimizing a loss function averaged over the training data, such as the squared-error or a likelihood-based loss function

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^N L\left(y_i, \sum_{m=1}^M \beta_m b(x_i; \gamma_m)\right).$$

For many loss functions $L(y, f(x))$ and/or basis functions $b(x; \gamma)$, this requires computationally intensive numerical optimization techniques.

3.3 Feature selection schemes

Various feature selection methods (Zhao et al., 2019; Bolon-Canedo et al., 2013; Chandrashekar and Sahin, 2014; Tang et al., 2014) are available to reduce the entire feature set to a more compact one. Such methods can be roughly divided into three categories: filter methods, wrapper methods, and embedded methods. The advantage of the filter method is the computation efficiency and generalizability to different machine learning models. From the machine learning standpoint, the ideal filter method can effectively reduce the redundant features while keeping the relevant features for the model. It is known that the m best features could be not the best m features (Cover, 1974) because many important features are correlated and redundant. In this work we will be using three filtering feature selection schemes:

- $\langle \text{RRC} \rangle$ — “Redundancy Removed Correlated” features – is a scheme, when we (i) calculate Pearson linear correlation James et al. (2017) of all features with ground-truth labels; (ii) sort features according to increasing of their p -values and remove features (from the end of the obtained list) which accept the null hypothesis when their p -value is larger than 0.01, i.e., who meet the criterion that there is no relationship between X and Y ,”; and (iii) calculate the mutual pair-wise Pearson linear correlation among the remaining features and again remove features, which mutual correlations is bigger than some threshold (we will be using threshold 0.9 in this work).

- $\langle \text{RFRanks} \rangle$ — "Random Forest ranks" — is a scheme, when (i) we take $\langle \text{RRC} \rangle$ features from the previous step and (ii) we train RF model in a 5-fold cross-validation fashion, on each fold RF gives us ranks of selected features used for the classification for that fold, and (iii) for the final list, we select only the features which are intersect in the three folds of five.
- $\langle \text{mRMR} \rangle$ — "minimum Redundancy Maximum Relevance" — provides feature selection considering both the relevance for predicting the outcome variable and the redundancy within the selected features. The optimal characterization condition often means the minimal classification error. In an unsupervised situation where the classifiers are not specified, minimal error usually requires the maximal statistical dependency of the target class c on the data distribution in the subspace R^m (and vice versa). This scheme is maximal dependency (Max-Dependency). One of the most popular approaches to realize Max-Dependency is maximal relevance (Max-Relevance) feature selection: selecting the features with the highest relevance to the target class c . Relevance is usually characterized in terms of correlation or mutual information, of which the latter is one of the widely used measures to define dependency of variables. The $\langle \text{mRMR} \rangle$ method addresses mutual-information-based feature selection (Peng et al., 2005).

Assuming there are in total m features, and for a given feature X_i , $i \in \{1, \dots, m\}$, its feature importance based on the $\langle \text{mRMR} \rangle$ criterion can be expressed as

$$f^{mRMR}(X_i) = I(Y, X_i) - \frac{1}{|S|} \sum_{X_s \in S} I(X_s, X_i),$$

where Y is the response variable (class label), S is the set of selected features, $|S|$ is the size of the feature set (number of features), $X_s \in S$ is one feature out of the feature set S , X_i denotes a feature currently not selected: $X_i \notin S$. The function $I(\cdot, \cdot)$ is the mutual information

$$I(Y, X) = \int_{\Omega_Y} \int_{\Omega_X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) dx dy,$$

where Ω_Y and Ω_X are the sample spaces corresponding to Y and X , $p(x, y)$ is the joint probability density, and $p(\cdot)$ is the marginal density function. For discrete variables Y and X , the mutual information formula takes the form

$$I(Y, X) = \sum_{y \in \Omega_Y} \sum_{x \in \Omega_X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right).$$

In the <mRMR> feature selection process, at each step, the feature with the highest feature importance score $\max_{X_i \notin S} f^{mRMR}(X_i)$ will be added to the selected feature set S .

3.4 tSNE visualization algorithm

t-distributed stochastic neighbour embedding (t-SNE) is an unsupervised, non-linear technique used for data exploration and visualizing high-dimensional data (Van der Maaten and Hinton, 2008). It is based on the stochastic neighbour embedding (SNE) method proposed by Hinton and Roweis (2002). It minimizes a single Kullback-Leibler divergence between a joint probability distribution P in the high-dimensional space

and a joint probability distribution Q in the low-dimensional space, i.e., it minimizes

$$\text{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}},$$

where p_{ii} and q_{ii} are set to zero. This approach is called symmetric SNE, because it has the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for $\forall i, j$. In symmetric SNE, the pairwise similarities in the low-dimensional map q_{ij} are given by

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_k - y_l\|^2)}. \quad (3.2)$$

The way to define the pairwise similarities in the high-dimensional space p_{ij} is

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma^2)},$$

but this causes problems when a high-dimensional datapoint x_i is an outlier (i.e., all pairwise distances $\|x_i - x_j\|^2$ are large for x_i). For such an outlier, the values of p_{ij} are extremely small for all j , so the location of its low-dimensional map point y_i has very little effect on the cost function. As a result, the position of the map point is not well determined by the positions of the other map points. This problem is circumvented by defining the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is, we set

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}.$$

This ensures that

$$\sum_j p_{ij} > \frac{1}{2n}$$

for all datapoints x_i , as a result of which each datapoint x_i makes a significant contribution to the cost function. In the low-dimensional space, symmetric SNE simply uses (3.2). The main advantage of the symmetric SNE is the simpler form of its gradient, which is faster to compute. The gradient of symmetric SNE is given by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij}) (y_i - y_j).$$

Today, the tSNE method is a standard de-facto approach in visualizing high dimensional data (Silver et al., 2017).

Chapter 4

Experimental results

4.1 Hardware and software environment

4.1.1 Hardware and software

A computational server used in the experimental work is rented on the Amazon AWS EC2 services¹. The server has 8-core Intel(R) Xeon(R) Platinum 8259CL @ 2.50GHz processor and 32 GB of RAM. XGBoost models are trained in CPU mode on all 8 available cores. NVIDIA Tesla T4 GPU card² is used for training of all deep CNN models. Tesla T4 has NVIDIA Turing architecture, 2560 CUDA cores, 14 GB of the on-board memory and provides 8.1 TFLOPS in single-precision operations. The server is worked under Ubuntu 18.04 operating system.

The Matlab 2019b software³ is used for: (i) preparation of HDF5 version of the

¹<https://aws.amazon.com/>, accessed November 2020.

²<https://www.nvidia.com/en-us/data-center/tesla-t4/>, accessed November 2020.

³<https://www.mathworks.com/products/matlab.html>, accessed November 2020.

CIFAR10 dataset, (ii) implementation of <RRC>, <RFranks> and <mRMR> feature selection schemes, (iii) obtaining results from Caffe deep learning framework, (iv) extracting features from last layers of each CNN model, (v) running the tSNE visualization algorithm, and (vi) numerical calculation of classification results and confusion matrices. The Python library <sklearn>⁴ is used for running XGBoost classifier.

4.1.2 Caffe deep learning framework

State-of-the art deep learning frameworks nowadays include TensorFlow, PyTorch, Caffe, Deeplearning4j and others⁵. Caffe (Jia et al., 2014) is chosen to implement CNN models because this framework has the following advantages over others: (i) it is simple since it allows to describe a model of a deep network and its training parameters as text files (prototxt) and run models directly from a Linux operating system, (ii) it is more universal since it has Python and Matlab wrappers allowing to call trained Caffe models in these programming languages, and (iii) it is very fast because implemented on C++.

Some examples of how to describe layers of a deep network in Caffe are shown in Figure 4.1. The layer of the basic weighed sum operation entitled as `INNER_PRODUCT` and is shown in Figure 4.1(a). Description starts with identifiers `bottom` and `top`, which assign names of input and output variables `encode1` and `encode2` accordingly. The next identifier `name` describes name of the layer used by Caffe interpreter to connect layers among each other to build a deep network. The layer identifier

⁴https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn, accessed November 2020.

⁵<https://www.analyticsvidhya.com/blog/2019/03/deep-learning-frameworks-comparison>, accessed November 2020.

`type` describes the type of the layer, and an identifier `INNER_PRODUCT` explains to Caffe that this layer implements a weighed sum operation, i.e., input data located in the variable, `encode1` are multiplied by weights of neurons. Identifiers `blobs_lr` and `weight_decay` describe the initialization values for a learning rate and additional updating parameters for weights, for example to exponentially decay to zero at the training stage. Next, the weighted sum parameters are described in the section `inner_product_param`: the identifier `num_output` describes how many neurons will be in this layer, the identifiers `type`, `std` and `sparse` in the section `weight_filler` describe weights' initialization type as `gaussian` with the standard deviation of weights' distribution `1` and weights' sparsification (sparseness) parameter `15`. Similar parameters can also be chosen for biases at the end of the layer description. The activation function layer in Figure 4.1(b) has identifiers `bottom`, `top` and `name` which describe names of the input and output variables and name of the layer. An identifier `type`: `SIGMOID` shows that this layer implements a sigmoid activation function of neurons. The use of same input and output variable `encode2` means that the layer takes values from that variable, converts them by sigmoid and saves them back into the same variable. The accuracy layer in Figure 4.1(c) calculates classification accuracy between two input variables `encode4` and `label` and saves the result into output variable `accuracy` for both `TRAIN` and `TEST` phases. The softmax loss layer in Figure 4.1(d) calculates the loss between two input variables `encode4` and `label` and saves the result into output variable `loss`.

The learning parameters of the CNN are defined in the solver `<prototxt>` file (Figure 4.2). The solver file consists of three sections. The first section contains a number of patterns on which the trained model will be tested (calculated as a

```

layers {
  bottom: "encode1"
  top: "encode2"
  name: "encode2"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 1
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 180
    weight_filler {
      type: "gaussian"
      std: 1
      sparse: 15
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
a)

layers {
  bottom: "encode2"
  top: "encode2"
  name: "encode2"
  type: SIGMOID
}
b)

layers {
  name: "accuracy"
  type: ACCURACY
  bottom: "encode4"
  bottom: "label"
  top: "accuracy"
  include: { phase: TRAIN }
  include: { phase: TEST }
}
c)

layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "encode4"
  bottom: "label"
  top: "loss"
}
d)

```

Figure 4.1: Descriptive examples of layers in Caffe.

total number of testing patterns in dataset divided by size of a training batch), and a number of training iterations at which that testing should be done. The second section contains type of a solver to use, a base learning rate, momentum, and a weight decay of the network as well as a policy to change a learning rate. Caffe supports several solvers — methods to address the general optimization problem of loss minimization — such as SGD, AdaDelta, adaptive gradient, Nesterov’s accelerated gradient and others. Decreasing learning rate as an optimization/learning process normally helps obtain better results. In our example in Figure 4.2 we have a “fixed” learning rate policy, i.e., the learning rate does not change during training. Several policies are allowable, including:

- **step** (learning rate decreases by some step value every designated number of training iterations),
- **multistep** (learning rate decreases by some step on some specific training iteration(s)),
- **inv** (inverse decay - it changes inversely to a number of training iterations).

Momentum and weight-decay are other parameters to change during an optimization process. The last section of the solver file contains the maximum number of training iterations, the number of iterations when the model should save training and testing results, and a solver mode which specifies what hardware CPU or GPU to use for training. More information about Caffe deep learning framework is available at the official page⁶.

4.2 CIFAR10 dataset

4.2.1 Original data

The CIFAR10 dataset consists of 60,000 32×32 colour images of 10 classes, with 6,000 images per class (Krizhevsky, 2009). Classes are: "Airplane", "Automobile", "Bird", "Cat", "Deer", "Dog", "Frog", "Horse", "Ship" and "Truck". There are 50,000 training images and 10,000 test images. The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains 1,000 randomly-selected images from each class. Training batches contain the remaining images in random order, but some training batches may contain more images from one

⁶<https://caffe.berkeleyvision.org>, accessed November 2020.


```
net: "cifarTrTe.prototxt"
# test_iter specifies how many forward test iterations should be
# carried out. If we have test batch size 100 and 100 test
# iterations, we cover all 10,000 testing images.
test_iter: 100
# Carry out testing every 2000 training iterations
test_interval: 2000

# Solver, the base learning rate, momentum and weight decay
type: SGD
base_lr: 0.001
momentum: 0.9
weight_decay: 0.004
# The learning rate policy
lr_policy: "fixed"

# Display every 2000 iterations
display: 2000
# The maximum number of iterations
max_iter: 80000
# snapshot intermediate results
snapshot: 2000
snapshot_prefix: "cifar_m01"
# solver mode: CPU or GPU
solver_mode: GPU
```

Figure 4.2: Example of the solver file in Caffe.

class than another. Classes are completely mutually exclusive. There is no overlap between automobiles and trucks. Class "Automobile" includes sedans, SUVs and others, and class "Truck" includes only big trucks. Data preparation stage included datasets creation for the CNN and XGBoost parts.

Creation of the dataset for the CNN part. Caffe reads data through data layers⁷ which support LevelDB and LMDB databases, common image formats (.jpg, .png, etc.) and HDF5 (hierarchical data format). All working datasets were created using the HDF5 format because it provides efficient data storage and data organization on logical level. Because original data are stored in a form of data entries, it was

⁷<https://caffe.berkeleyvision.org/tutorial/layers.html#data-layers>, accessed November 2020.

necessary to convert them properly into a form of 2D images. Each data entry was a row of a matrix with 3,072 elements which stores a 32×32 -pixel colour RGB image. The first 1,024 elements contain the red channel values, the next 1,024 the green, and the final 1,024 entries contain the blue channel values. For the Caffe-compatible HDF5-type dataset we form data as a matrix with a shape

$$\langle \text{height} \rangle \times \langle \text{width} \rangle \times \langle 3 \rangle \times \langle N \rangle,$$

where $\langle \text{height} \rangle \times \langle \text{width} \rangle$ are numbers of pixels of an image, $\langle 3 \rangle$ is a number of color channels (red, green, blue) and $\langle N \rangle$ is a number of images. During the training set creation we also created the mean image to be used for data normalization⁸. The normalization is done by (i) subtracting the mean image from every image in the training set and (ii) by the following z -scoring every pixel within each channel, so all pixel values within each color channel are centered to have mean 0 and scaled to have standard deviation 1. That normalization is an essential step which allows all images to be in the same conditions. Training and testing datasets were created independently by different scripts and saved into different files to avoid any possible data contamination. During creation of datasets, a random check for the image-label correspondence was done (Figure 4.3) ensuring their correctness as well as final print of the obtained HDF5 structure (Table 4.1).

Creation of the dataset for the XGBoost part. In this work features learned by the CNN part and available on the last output layer of CNN are used for training the XGBoost model. For each of the experiments below, an appropriately trained CNN model was run over original training and testing datasets. Then, obtained

⁸<http://cs231n.github.io/linear-classify/>, accessed November 2020.

Table 4.1: Output HDF5 structures of CIFAR10 training and testing datasets.

Structure of HDF5 dataset for training	Structure of HDF5 dataset for testing
HDF5 cifar10TR.h5 Group '/' Dataset 'data' Size: 32x32x3x50000 MaxSize: 32x32x3x50000 Datatype: H5T_IEEE_F32LE (single) ChunkSize: [] Filters: none FillValue: 0.000000 Dataset 'label' Size: 1x50000 MaxSize: 1x50000 Datatype: H5T_IEEE_F32LE (single) ChunkSize: [] Filters: none FillValue: 0.000000	HDF5 cifar10TE.h5 Group '/' Dataset 'data' Size: 32x32x3x10000 MaxSize: 32x32x3x10000 Datatype: H5T_IEEE_F32LE (single) ChunkSize: [] Filters: none FillValue: 0.000000 Dataset 'label' Size: 1x10000 MaxSize: 1x10000 Datatype: H5T_IEEE_F32LE (single) ChunkSize: [] Filters: none FillValue: 0.000000

features (different shapes of 2D and 1D matrices depending on CNN architecture in different experiments) are taken from last or next-to-last layers of appropriate CNN and converted into a 2D array of $\langle n \times p \rangle$ form, where $\langle n \rangle$ is a number of observations (images) and $\langle p \rangle$ is a number of learned features. Depending on the experiment, it was done in a one-module fashion or 5-fold CV (cross-validation) fashion (we actually trained 5 CNN models). Also, the extracted features from the CNN part were saved as originally obtained values as well as normalized values. The following normalization schemes are used:

- (i) $\langle \text{submean} \rangle$, vector of columns mean values was substituted from each feature vector (feature-wise) to provide mean 0 for every feature column;
- (ii) $\langle \text{zscore} \rangle$, each feature column was centered to have mean 0 and scaled to have

standard deviation 1 (Goodfellow et al., 2016).

For each experiment, some preliminary runs were fulfilled and the best normalization scheme was used then to run XGBoost part of that experiment.



Figure 4.3: Random visual check of CIFAR10 classes while creating HDF5 training and testing datasets.

4.2.2 Augmented data

Data augmentation is used to add some distortion to the original CIFAR10 dataset with intention that this operation can improve the variance of output features extracted from the last layer of a CNN. Popular data augmentation approaches (Springenberg et al., 2014) normally include increasing image resolution as well; however, to compare two exact CNN architectures (number of input CNN planes should be equal as well), the original CIFAR10 resolution 32×32 was not changed. The example of an original image of a horse and 8 added distortions are shown in Figure 4.4. For each original image in the CIFAR10 dataset, such operations as shifting to left, right, top and bottom by 4 pixels, two rotations to the left by 10 and 20 degrees and two rotations to the right by 10 and 20 degrees are fulfilled. Only the training dataset was augmented, 8 distortions per image were applied, a total of 400,000 new images were

obtained. The size of a new data augmented training dataset of CIFAR10 was set to 450,000 images, 400000 distorted images plus 50,000 original images. The size and shape of the testing dataset remains unchanged to provide an apples-to-apples comparison of the results. Creation of working HDF5 data augmented training dataset for the CNN part as well as extracted features dataset for the XGBoost part were fulfilled in a similar way as described in the Section 4.2.1 above.



Figure 4.4: Example of data augmentation.

4.3 XGBoost results

Before experiments with combined CNN+XGBoost models, it is necessary to assess the accuracy of the XGBoost model. The raw CIFAR10 image has 32×32 pixels with 3 color channels, which makes it 3,072 features in total. A normalized version of the working HDF5 dataset with mean 0 and standard deviation 1 over all features (pixels) was used in the experiment. From a total of 3,072, 1,713 and 250 best features were selected by the $\langle \text{RRC} \rangle$ and $\langle \text{RFranks} \rangle$ feature selection schemes respectively. The XGBoost algorithm trained 164, 85 and 12 minutes for 3,072, 1,713 and 250 features, respectively, on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. We have obtained 50.10% of accuracy for the model trained on all 3,072 features, 49.29% for the model trained on 1713 $\langle \text{RRC} \rangle$ features, and 43.73% for the model trained on 250 $\langle \text{RFranks} \rangle$ features.

4.4 AlexNet+XGBoost results

The AlexNet Caffe implementation is obtained from examples of original Caffe package (Krizhevsky, 2009). AlexNet is a famous deep CNN approach (Krizhevsky et al., 2012), it consists of 3 convolutional, 3 pooling, and 2 fully-connected layers (Table 4.2). Its graphical representation is depicted in Figure 4.5.

Table 4.2: AlexNex summary.

Architecture	3 convolutional, 3 pooling (1 MAX pooling, 2 AVE pooling) and 2 fully-connected layers, ReLU activation
Solver	type: SGD, base_lr: 0.001, momentum: 0.9, weight_decay: 0.004, lr_policy: "fixed"

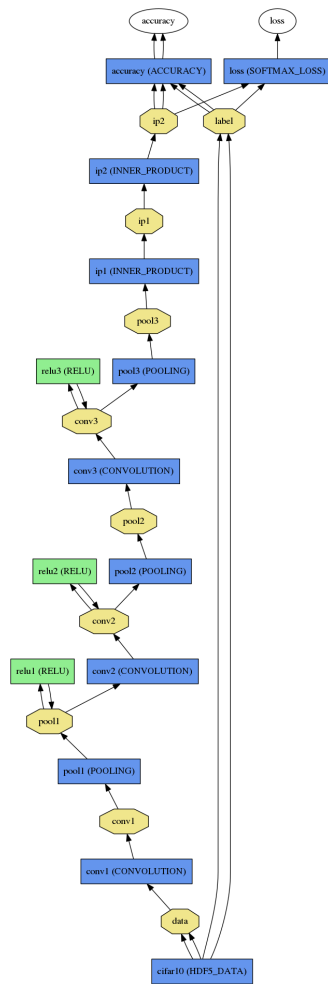


Figure 4.5: Graphical representation of AlexNet CNN.

Blue elements in Figure 4.5 represent layers of neurons where information is processing, yellow elements represent input and output matrices (blobs) associated with layers of neurons, and green elements represent activation functions associated with selected layers of neurons. The AlexNet architecture has 32 neurons (they are also called planes) in the first convolutional layer, 32 neurons in the second convolutional layer, 64 neurons in the third convolutional layer, 64 neurons in the fourth fully-connected layer, and 10 output fully-connected neurons which correspond to 10

labels of CIFAR-10 dataset. ReLU are used as an activation function. The loss values for the training and testing stages and classification accuracy are calculated on the “loss” and “accuracy” layers.

The declared accuracy of AlexNet on the CIFAR10 test set in the field is around 75% (Krizhevsky et al., 2012) and it has been effectively reached in our experiments (74.71%). AlexNet was trained up to 80,000 iterations, which took about one hour on GPU Tesla T4. A small modification in the architecture has been made: original AlexNet CNN has 64 output neurons in the next-to-last fully-connected layer. It was increased to 256 output neurons to extract more features from the CNN part to train XGBoost part. Preliminary research showed that the < submean > extracted feature normalization scheme given slightly better performance for the XGBoost part. From a total of 256, 239 and 150 best features were selected by < RRC > and < RFranks > feature selection schemes, respectively. The best XGBoost result was selected from a set of experiments with changing a number of trees from 100 to 1800 within three feature selection schemes: (i) all extracted 256 features, (ii) 239 features selected by < RRC >, and (iii) 150 features selected by < RFranks >. The XGBoost part experiment took from 1 to 47 minutes per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. Results using all extracted 256 features are presented in Table 4.3.

Table 4.3: XGBoost accuracy using 256 < submean > AlexNet features.

NumTrees	100	200	300	600	900	1200	1500	1800
Accuracy, %	73.00	–	75.35	75.78	76.18	76.11	76.19	76.34

4.5 All Convolutional Net + XGBoost results

The All Convolutional Net (AllConvNet) Caffe implementation is obtained from GitHub repository of Mateus Zbuda⁹. AllConvNet is a famous deep CNN model proposed in Springenberg et al. (2014). It consists of 9 convolutional, 2 dropout and 1 pooling layers, and ReLU units are used as an activation function (Table 4.4).

Table 4.4: AllConvNet summary.

Architecture	9 convolutional, 2 dropout, 1 pooling (AVE), ReLU activation
Solver	type: SGD, base_lr: 0.05, momentum: 0.9, weight_decay: 0.001, lr_policy: "multistep", gamma: 0.1

The declared accuracy of the used Caffe AllConvNet implementation on CIFAR10 test set is 90.25%⁹. This implementation does not deal with any augmentation approach. The original paper (Springenberg et al., 2014) declares 95.59% and this result was obtained by using additional augmentation of the training data set, when each image was enlarged to 224×224 pixels and slightly changed by left-right rotations and shifting. The declared level of accuracy was not reached in our experiments (83.54%). The AllConvNet model was trained up to 80,000 iterations, which took about 7 hours on GPU Tesla T4. An accuracy of 90.25% was not reached and that is probably because that was gained by applying global contrast normalization and ZCA (zero-phase component analysis) whitening⁹. An accuracy of 95.59% was not reached because the model was not trained on the augmented dataset.

In all, 64 CNN features were extracted from the last CNN layer to train the XGBoost part. Preliminary research showed that the original features with no normalization gives slightly better performance for the XGBoost part. We did not run

⁹<https://github.com/mateuszbeda/ALL-CNN>, accessed November 2020.

< RRC > and < RFranks > feature selection schemes in this experiment because 64 features is already a small number and there is no room to decrease this number further. The best XGBoost result was selected from a set of experiments with changing the number of trees from 100 to 1,800, which took from 1 to 5 minutes per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. The XGBoost part results using all extracted 64 features are presented in Table 4.5.

Table 4.5: XGBoost accuracy using 64 <original> AllConvNet features.

NumTrees	100	200	300	600	900	1200	1500	1800
Accuracy, %	84.16	84.04	84.06	83.98	84.02	84.05	84.01	84.01

4.6 WideResNet + XGBoost results

WideResNet Caffe implementation is obtained from GitHub repository of user Revilokeb¹⁰. WideResNet is a famous deep CNN model proposed in Zagoruyko and Komodakis (2016). It is a huge network and it consists of 29 convolutional, 23 batch normalization, 13 eltwise, 23 scaling, 1 pooling and 1 fully-connected layers. ReLU units are used as an activation function (Table 4.6).

Table 4.6: WideResNet summary.

Architecture	29 convolutional, 23 batch normalization, 13 eltwise, 23 scaling and 1 pooling (AVE), 1 fully-connected, ReLU activation
Solver	type: Nesterov, average_loss: 400, base_lr: 0.1, momentum: 0.9, weight_decay: 0.0005, lr_policy: "step", gamma: 0.2

The declared accuracy of the used Caffe WideResNet implementation on CIFAR10

¹⁰https://github.com/revilokeb/wide_residual_nets_caffe, accessed November 2020.

test set is 92.54%¹⁰ without applying global contrast normalization and ZCA whitening. The original paper (Zagoruyko and Komodakis, 2016) declares 95.83% with padding and applying global contrast normalization and ZCA whitening. The declared levels of accuracy have not been reached in our experiments (86.13%) due to (i) less time the model was trained and (ii) padding and global contrast normalization and ZCA whitening were not applied as in the original paper (i.e., Zagoruyko and Komodakis, 2016). The WideResNet model was trained up to 80,000 iterations, which took about 23 hours on GPU Tesla T4. WideResNet has 640 output neurons in the last convolutional layer and 10 output neurons in the last fully-connected layer. Therefore, 640 features from the CNN part were extracted to train the XGBoost part. Preliminary research showed that the `< zscore >` extracted feature normalization scheme gives slightly better performance for the XGBoost part. From a total of 640, 618 and 250 best features were selected by `< RRC >` and `< RFranks >` feature selection schemes, respectively. The best XGBoost result was selected from a set of experiments with changing a number of trees from 100 to 1,800 within three feature selection schemes: (i) all extracted 640 features, (ii) 618 features selected by `< RRC >`, and (iii) 250 features selected by `< RFranks >`. The XGBoost part experiment took from 4 to 20 minutes per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. The XGBoost part results using all extracted 640 features are presented in Table 4.7.

Table 4.7: XGBoost accuracy using 640 `<original>` WideResNet features.

NumTrees	100	200	300	600	900	1200	1500	1800
Accuracy, %	85.61	85.67	85.63	85.63	85.63	85.63	85.63	85.63

4.7 DenseNet + XGBoost results on original data

DenseNet Caffe implementation is obtained from GitHub repository of user shen865069799¹¹. DenseNet is a famous deep CNN model proposed in Huang et al. (2016). It is a huge network and it consists of 39 convolutional, 39 batch normalization, 39 scaling, 1 pooling and 1 fully-connected layers. ReLU units are used as an activation function (Table 4.8).

Table 4.8: DenseNet summary.

Architecture	39 convolutional, 39 batch normalization, 39 scaling and 1 pooling (AVE), 1 fully-connected, ReLU activation
Solver	type: Nesterov, base_lr: 0.1, momentum: 0.9, weight_decay: 0.0001, lr_policy: "multistep", gamma: 0.1

The declared accuracy of the used Caffe DenseNet implementation on CIFAR10 test set is 92.91%¹¹. The original paper (Huang et al., 2016) declares 94.81% within the Torch deep learning framework implementation. The declared levels of accuracy have not been reached in our experiments (90.23%) due to (i) less time we trained the model and (ii) technical differences in Caffe and Torch deep learning frameworks. The DenseNet model was trained up to 230,000 iterations, which took about 16 hours on GPU Tesla T4. DenseNet has 448 output neurons in the last convolutional layer and 10 output neurons in the last fully-connected layer. Therefore, 448 features from the CNN part were extracted to train the XGBoost part. Preliminary research showed that the < submean > feature normalization scheme given slightly better performance for the XGBoost part. From total 448, 430 and 150 best features were selected by < RRC > and < RFranks > feature selection schemes, respectively. The best XGBoost result was selected from a set of experiments with changing a number

¹¹<https://github.com/shen865069799/DenseNetCaffe>, accessed November 2020.

of trees from 100 to 1,800 within three feature selection schemes: (i) all extracted 448 features, (ii) 430 features selected by $\langle \text{RRC} \rangle$, and (iii) 150 features selected by $\langle \text{RFranks} \rangle$. The XGBoost part experiment took from 4 to 16 minutes per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. The XGBoost part results using all extracted 448 features are presented in Table 4.9.

Table 4.9: XGBoost accuracy using 448 $\langle \text{submean} \rangle$ DenseNet features.

NumTrees	100	200	300	600	900	1200	1500	1800
Accuracy, %	89.86	89.98	90.09	90.17	90.16	90.14	90.13	90.13

When the classification accuracy on the training set of 50,000 CIFAR10 images was checked in previous experiments, it was noted that it equals to 100% due to the obvious over-fitting of the CNN part to the training dataset. Therefore, it is expedient to run 5-fold CV experiment to check if non-overfitted models can provide better feature extraction from the CNN part. The best CNN part model, the DenseNet, was selected: each of 5-fold CV CNNs is trained on 40,000 examples of its own fold of the training set and generated features both for (i) its own testing fold of 10000 images of the regular training set as well as (ii) for regular 10000 testing set of CIFAR10. Five DenseNet models were trained up to 200000 iterations each, which took about 60 hours on GPU GTX Tesla T4. DenseNet trained in a 5-fold CV fashion outperformed the one-module model and provided 91.31% classification accuracy instead of 90.23%. Similarly, 448 CNN features from the CNN part were extracted to train the XGBoost part. Preliminary research showed that the $\langle \text{zscore} \rangle$ extracted feature normalization scheme gives slightly better performance for the XGBoost part. Similarly, 430 and 150 best features were selected by $\langle \text{RRC} \rangle$ and $\langle \text{RFranks} \rangle$ feature selection schemes, respectively. The best XGBoost result was selected from a

set of experiments with changing a number of trees from 100 to 1,800 within all four feature selection schemes: (i) all extracted 448 features, (ii) 338 features selected by $\langle \text{RRC} \rangle$, (iii) 150 features selected by $\langle \text{RFranks} \rangle$, and (iv) 313 features selected by $\langle \text{mRMR} \rangle$. The XGBoost part experiment took from 2 to 46 minutes per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. The XGBoost part results using all feature selection schemes are presented in Table 4.10. We provided results of all XGBoost runs for this model, DenseNet, and for this 5-fold CV experiment only, since XGBoost behaviour was similar to the presented in all other models and experiments.

Table 4.10: XGBoost accuracy using all feature selection schemes for DenseNet features obtained in 5-fold CV fashion.

NumTrees	100	200	300	600	900	1200	1500	1800
448 all features accuracy, %	90.08	n/a	90.21	90.51	90.65	90.66	90.71	90.62
338 $\langle \text{RRC} \rangle$ features accuracy, %	90.16	-	90.24	90.22	90.36	90.33	90.41	90.29
150 $\langle \text{RFranks} \rangle$ features accuracy, %	90.23	90.19	90.15	90.22	90.29	90.36	90.42	90.39
313 $\langle \text{mRMR} \rangle$ features accuracy, %	89.90	90.14	90.21	90.37	90.47	90.47	90.46	90.58

4.8 DenseNet + XGBoost results on augmented data

Exactly the same DenseNet model architecture, training parameters and feature selection schemes described in Section 4.7 were used to train DenseNet on the augmented training data set and to test it on the regular CIFAR10 test set. Comparing with the regular dataset, better classification results were obtained on the augmented dataset

for the CNN part: 91.66% accuracy on the augmented versus 90.23% on the regular for one-module fashion, and 92.64% accuracy on the augmented versus 91.31% on the regular for the 5-fold CV fashion. Due to the 9-times increased data volume in the augmented dataset (it was increased from 50000 to 450000 images), all training times for both CNN and XGBoost parts were increased significantly. DenseNet model was trained up to 230,000 iterations, which took about 21 hours on GPU Tesla T4 for one-module fashion and 90 hours for the 5-fold CV fashion. The XGBoost part experiment took from 2 to 7 hours per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. Therefore, Table 4.11 does not contain all cells filled. The best XGBoost result was selected from a set of experiments with changing a number of trees from 100 to 1800 within three feature selection schemes: (i) all extracted 448 features, (ii) 430 features selected by < RRC >, and (iii) 150 features selected by <RFranks>. The XGBoost part results for both one-module and 5-fold CV runs are presented in Table 4.11.

Table 4.11: XGBoost accuracy using 448 <submean> features extracted by DenseNet on augmented dataset in one-module and 5-fold CV fashions.

NumTrees	100	200	300	600	900	1200	1500	1800
One-module, all 448 features accuracy, %	90.98	-	91.55	91.54	91.54	91.53	-	-
5-fold CV, <RFranks> 150 features accuracy, %	91.03	-	91.44	91.48	-	-	-	-

4.9 Transfer Learning: CaffeNet pre-trained on ImageNet dataset + XGBoost results

CaffeNet Caffe implementation is obtained from examples of original Caffe package¹². CaffeNet is a single-GPU version of AlexNet CNN which won the 2012 ILSVRC challenge with the validation error rate of 18.2% (Krizhevsky et al., 2012). CaffeNet consists of 5 convolutional, 3 pooling, 2 LRN, 2 dropout, and 3 fully-connected layers. ReLU units are used as an activation function (Table 4.12). Within the competition, CaffeNet was trained on ImageNet dataset with 1,000 classes, therefore its last fully-connected layer contains 1,000 neurons. To apply it for our case of CIFAR10, a little network “surgery” was done: 1,000 neurons of the last layer were substituted by new 248 neurons (their weight initialization started from 0) and it becomes the next to last layer, and the last layer with 10 neurons corresponding to a number of CIFAR10 classes was added. CIFAR10 dataset images were enlarged from 32×32 to 227×227 pixels to provide proper fine-tuning. Weight coefficients for all layers except the two last layers were copied from the original CaffeNet, and the new CaffeNet were fine-tuned by training on the CIFAR10 dataset for up to 120,000 training iterations, which took about 5 hours on GPU Tesla T4. During the fine-tuning process, the learning rate for all layers was chosen really small (see Table 4.12), and it was multiplied by 10 for the last two new layers ensuring decreasing of the training loss function for the new CaffeNet. We were able to achieve 86.63% of accuracy on the CIFAR10 test set with this approach.

¹²<https://caffe.berkeleyvision.org/gathered/examples/cifar10.html>, accessed November 2020.

Table 4.12: CaffeNet summary.

Architecture	5 convolutional, 3 pooling (MAX), 2 LRN, 2 dropout, 3 fully-connected, ReLU activation
Solver	type: type: Nesterov, base_lr: 0.001, momentum: 0.9, weight_decay: 0.0005, lr_policy: "multistep", gamma: 0.5

New CaffeNet 248 features from the CNN part were extracted to train the XGBoost part. Preliminary research showed that the < submean > feature normalization scheme given slightly better performance for the XGBoost part. From a total of 248, 200 and 150 best features were selected by < RRC > and < RFranks > feature selection schemes, respectively. The best XGBoost result was selected from a set of experiments with changing a number of trees from 100 to 1800 within three feature selection schemes: (i) all extracted 248 features, (ii) 200 features selected by < RRC >, and (iii) 150 features selected by < RFranks >. The XGBoost part experiment took from 4 to 49 minutes per run on 8 cores of Intel(R) Xeon(R) Platinum 8259CL. The XGBoost part results using 200 < RRC > features are presented in Table 4.13.

Table 4.13: XGBoost accuracy using 200 <submean> <RRC> CaffeNet features.

NumTrees	100	200	300	600	900	1200	1500	1800
Accuracy, %	85.64	85.79	85.79	85.79	85.79	85.79	85.79	85.79

4.10 Analysis and summary of results

All tSNE feature visualization results for the CIFAR10 training and testing sets are collected in Table 4.14. All top classification accuracies for all experiments are collected in Table 4.15. The cases where the XGBoost part outperforms the CNN part are highlighted in bold, and the cases where the CNN part outperforms the XGBoost

part are highlighted in italics. All confusion matrices for all experiments are collected in Appendix A. As was mentioned before, tSNE visualizations (Van der Maaten and Hinton, 2008) help to visually appraise a quality of features learned by the CNN part and avoid possible mistakes. As we can see from both Tables 4.14 and 4.15, our visualization results correspond to the values of classification accuracy, i.e., the CNN part with better classification accuracy shows better arrangement of learned features into 10 clusters, we better see groups of data and those groups are better separated from each other. In other words, a level of quality of features extracted by the CNN part could numerically be described by a value of classification accuracy of the appropriate CNN we extracted features from. Therefore, in the following conclusions, when we say “feature extraction quality from the CNN part has the accuracy 85%”, we mean that CNN with that learned features provides the classification accuracy of 85%. We can derive the following conclusions from obtained results:

1. The single deep CNN model significantly outperforms single XGBoost model for image classification tasks, i.e., 74.71% versus 50.1% on the CIFAR10 image dataset.
2. In AlexNet+XGBoost and AllConvNet+XGBoost experiments, when feature extraction quality from the CNN part has the accuracy less than 85%, the XGBoost part provides an improvement over the CNN part. In WideResNet+XGBoost, Densenet+XGBoost and CaffeNet+XGBoost experiments, when feature extraction quality from the CNN part has the accuracy more than 85%, the XGBoost part does not provide an improvement over the CNN part. That might be explained by a hypothesis, that at the lower quality of CNN feature

extraction, XGBoost model could train its weak learners better and their ensembling can find some missing information in the originally extracted CNN features and improve the final classification result. However, this opportunity vanishes, when a quality of CNN feature extraction is high enough.

3. CNN feature extraction fulfilled in 5-fold CV fashion provides better quality than one-module network: 91.31% versus 90.23% classification accuracy in DenseNet+XGBoost experiment on the original CIFAR10 dataset and 92.64% versus 91.66% classification accuracy in DenseNet+XGBoost experiment on the augmented CIFAR10 dataset. It could be due to the lower degree of overfitting of 5-fold CV models as well as due to the ensembling of the classification results by 5-fold CV models at the feature extraction stage.
4. Data augmentation improved a quality of CNN feature extraction by 1.3% as appropriate classification accuracies increased from 91.31% to 92.64% on the example of the CIFAR10 image dataset.
5. Used filtering feature selection methods \langle RRC \rangle , \langle RFranks \rangle , and \langle mRMR \rangle do not help improving XGBoost performance in comparison with originally generated number of features. It is probably due to the fact that CNN still preserves some degree of spatial information between the extracted features and removing some elements of that information deteriorates classification performance of the XGBoost part.
6. Experimental results obtained in this work correspond to the results reported by other researchers Ponomareva et al. (2017); Ren et al. (2017); Wan et al. (2020).

However, very interesting conclusion 2 about the 85% feature quality threshold is drawn from results provided by different CNN models, which might influence the threshold in different ways due to differences in their architectures and generalized abilities. Therefore, it is expedient to confirm this conclusion by using exactly the same CNN model at different values of classification accuracies, i.e., different qualities of feature extraction. In this case, it will be a fair apples-to-apples comparison. To fulfill this comparison, we have chosen Densenet+XGBoost one-module experiment performed on original CIFAR10 data (50,000 images for training, 10,000 images for testing). The network was re-trained with possibility to save more trained models (snapshots with intermediate results) with smaller step to cover the range 60–90% of feature extraction quality. We covered 60–80% with step 5% and 80–90% with step 1%. The real CNN accuracy was slightly different from feature extraction quality bin value, for example for bin value 60%, the accuracy was 60.64% and so on. At each step, features from the CNN part were extracted and XGBoost model was trained and provided appropriate predictions. We run 3 scenarios when we changed number of XGBoost trees as 50, 100, and 300. The results are presented in Figure 4.6 and Figure 4.7.

As we can see, for the Densenet+XGBoost experiment that threshold is a bit higher and equals 88%. Also Figure 4.6 shows how the improvement of feature quality extraction from the CNN part degrades the XGBoost part accuracy improvement. For example, for the XGBoost 50 trees scenario:

- at the CNN part accuracy 60.64%, XGBoost *considerably improved* classification accuracy to 79.58% (18.94% improvement);
- at the CNN part accuracy 80.09%, XGBoost *just slightly improved* classification

accuracy to 84.50% (4.41% improvement);

- at the CNN part accuracy 88.03%, XGBoost *did not improve* the final classification accuracy *at all*, its deteriorated to 87.80% (0.23% deterioration);
- at the CNN part accuracy 89% and more, XGBoost *did not improve* the final classification accuracy *anymore* and the accuracy deterioration is about 2%.

This observation is also true for two other scenarios, i.e., XGBoost 100 trees and XGBoost 300 trees (see Figure 4.6). Moreover, XGBoost spends much more time on the training process when it is training on the lower quality of CNN feature extraction, see XGBoost 300 trees scenario in Figure 4.7. It shows that a loss optimization process for weak learners of the XGBoost model needs much more computational time in case when CNN provided low quality features and it is much faster when the CNN features are “well” prepared.

Therefore, experimental results in Figures 4.6 and 4.7 have confirmed our hypothesis in the conclusion 2: “at the lower quality of CNN feature extraction, XGBoost model could train its weak learners better and their ensembling can find some missing information in the originally extracted CNN features and improve the final classification result”.

Table 4.14: tSNE visualizations of extracted features from CNN parts.

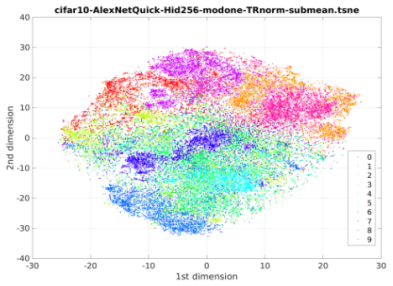
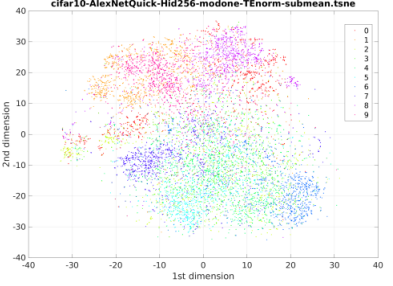
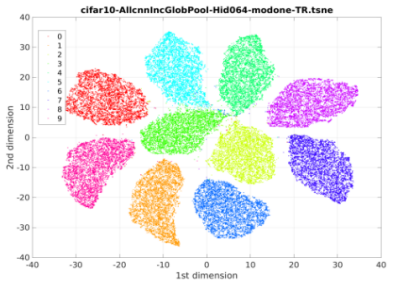
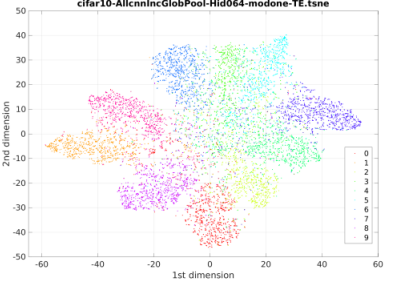
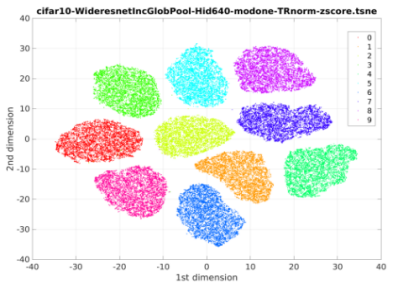
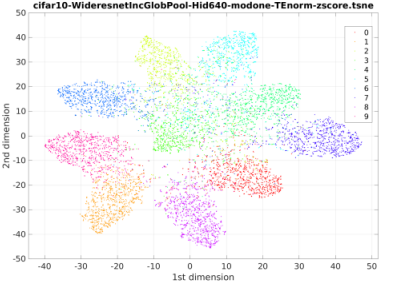
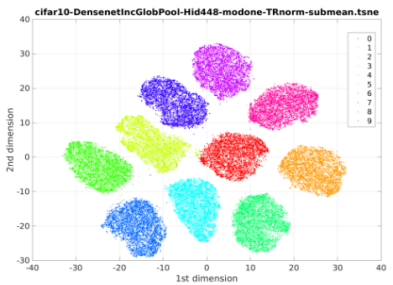
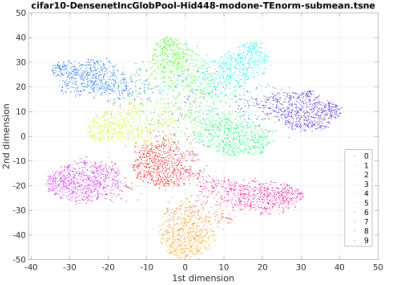
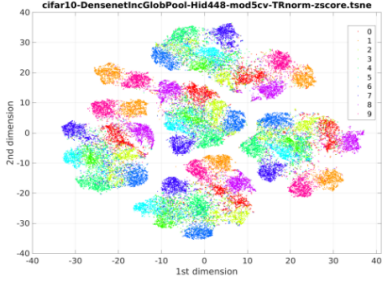
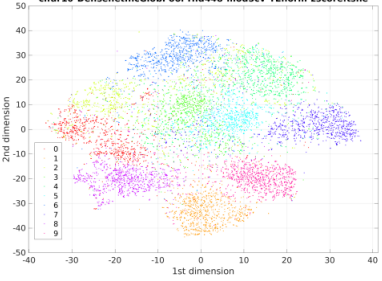
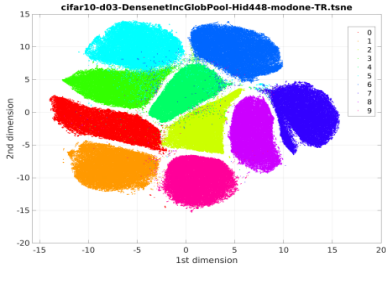
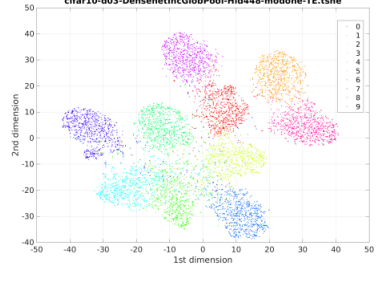
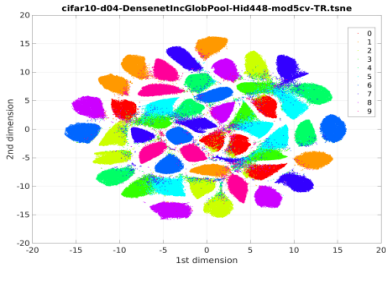
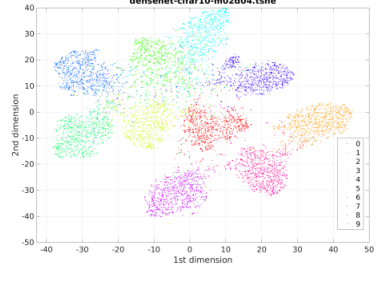
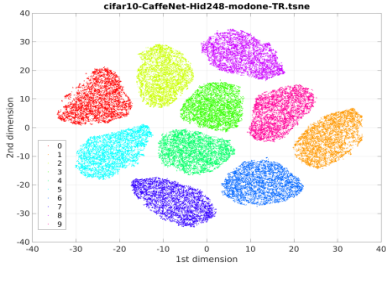
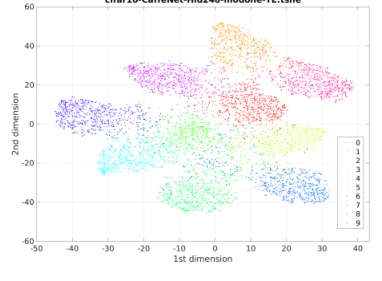
Model	Training set	Testing set
AlexNet		
AllConv Net		
WideRes Net		
DenseNet, one- module, original data		

Table 4.14 continued from previous page

Model	Training set	Testing set
DenseNet, 5-fold CV, original data		
DenseNet, one- module, augmented data		
DenseNet, 5-fold CV, augmented data		
CaffeNet, transfer learning		

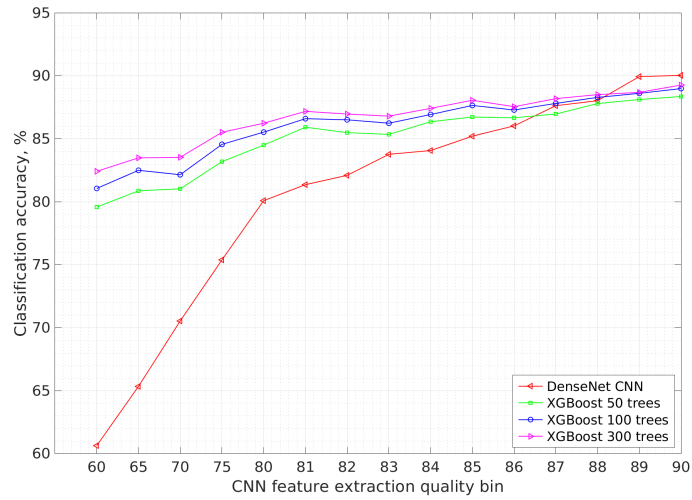


Figure 4.6: DenseNet vs XGBoost classification accuracy at different feature extraction quality.

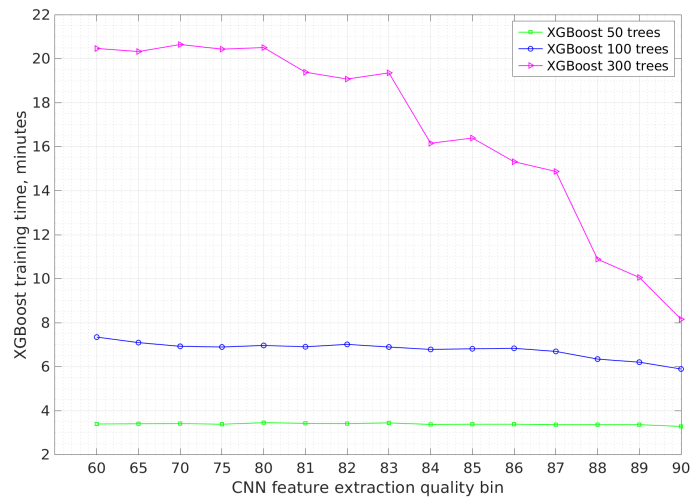


Figure 4.7: XGBoost training time at different feature extraction quality.

Table 4.15: Summary of classification results.

Model	Accuracy, %			
	CNN at feature extraction (accur / feat train time)	XGBoost using all features (accur / feat)	XGBoost using <RRC> features (accur / feat)	XGBoost using <RFranks> features (accur / feat)
AlexNet, one-module, Appendix A.1	74.71 / 256 1 hour	76.34 / 256	75.99 / 239	74.80 / 150
AllConvNet, one-module, Appendix A.2	83.54 / 640 7 hours	84.16 / 640	n/a	n/a
WideResNet, one-module, Appendix A.3	<i>86.13</i> / 640 23 hours	<i>85.67</i> / 640	85.68 / 618	85.65 / 250
DenseNet, one-module, Appendix A.4	<i>90.23</i> / 448 16 hours	<i>90.17</i> / 448	90.11 / 430	90.17 / 150
DenseNet, 5-fold CV, Appendix A.5	<i>91.31</i> / 448 60 hours	<i>90.71</i> / 448	90.41 / 383	90.42 / 150
DenseNet, one-module, data augmented, Appendix A.6	<i>91.66</i> / 448 21 hours	<i>91.55</i> / 448	91.54 / 383	91.54 / 150
DenseNet, 5-fold CV, data augmented, Appendix A.7	<i>92.64</i> / 448 90 hours	<i>91.37</i> / 448	91.18 / 383	91.48 / 150
CaffeNet transfer learning, one-module, Appendix A.8	<i>86.63</i> / 248 5 hours	<i>85.65</i> / 248	85.79 / 200	85.65 / 150

Chapter 5

Conclusions

The CNN+XGBoost combined model where learned representations extracted from the CNN part are used as input features for the XGBoost part is implemented and researched in this work. Obtained results of experimental research allow us to conclude the following:

1. A single deep CNN model significantly outperforms single XGBoost model for image classification tasks on the CIFAR10 image dataset (74.71% versus 50.1% classification accuracy).
2. When the CNN part has classification accuracy less than around 85–88%, features learned at this level of classification accuracy allow the XGBoost part to provide an improvement over the CNN part. In the opposite situation, the XGBoost part does not provide an improvement over the CNN part. That might be explained by the hypothesis that

at the lower quality of CNN feature extraction, the XGBoost model could train its weak learners better and their ensembling can

find some missing information in the originally extracted CNN features and improve the final classification result.

However, this opportunity vanishes, when a quality of CNN feature extraction is high enough.

3. CNN feature extraction fulfilled via 5-fold CV fashion provides better quality than a one-module network — 91.31% versus 90.23% classification accuracy in the DenseNet+XGBoost experiment on the original CIFAR10 dataset and 92.64% versus 91.66% of classification accuracy in the DenseNet+XGBoost experiment on the augmented CIFAR10 dataset. It could be due to the lower degree of overfitting of 5-fold CV models as well as due to the ensembling of classification results over 5-fold CV models at the feature extraction stage.
4. Data augmentation improved the quality of CNN feature extraction by 1.3 percentage points as appropriate classification accuracies increased from 91.31% to 92.64% on the example of the CIFAR10 image dataset.
5. The filtering feature selection methods used, i.e., $\langle \text{RRC} \rangle$, $\langle \text{RFranks} \rangle$ and $\langle \text{mRMR} \rangle$ do not help improve XGBoost performance in comparison with the originally generated number of features. It is probably due to the fact that CNN still preserves some degree of spatial information between the extracted features and removing some elements of that information deteriorates classification performance of the XGBoost part.

Experimental results obtained in this work correspond to results reported by other researchers (Ponomareva et al., 2017; Ren et al., 2017; Wan et al., 2020). However,

these literature results are contradictory: Ren et al. (2017) report that XGBoost allows to improve CNN accuracy and Wan et al. (2020) report the opposite conclusion that tree-based models did not outperform CNN. Also, to the best of our knowledge, there are not many similar research results published in the literature yet. The contribution of this work consists in taking one more step towards answering the question: could tree-based algorithms improve or outperform CNNs on image classification tasks? We answered that question by extensive experimental research using different CNN architectures which provide different levels of classification accuracies and, therefore, different qualities of features extracted by the CNN part. We found a threshold of feature quality extraction experimentally, when an appropriate CNN gives around 85–88% classification accuracy, to be a margin, after which a tree-based method, in particularly XGBoost, does not improve the final CNN+XGBoost classification accuracy anymore. We draw this conclusion based on experimental results obtained on the CIFAR10 image dataset.

Future research work could include two research directions at least: (i) instead of XGBoost use another tree-based method, in particular random forests, and (ii) instead of filtering feature selection approaches, used in this work, use wrapper and/or embedded methods of feature selection for the tree-based part.

Appendix A

Confusion matrices for all classification results

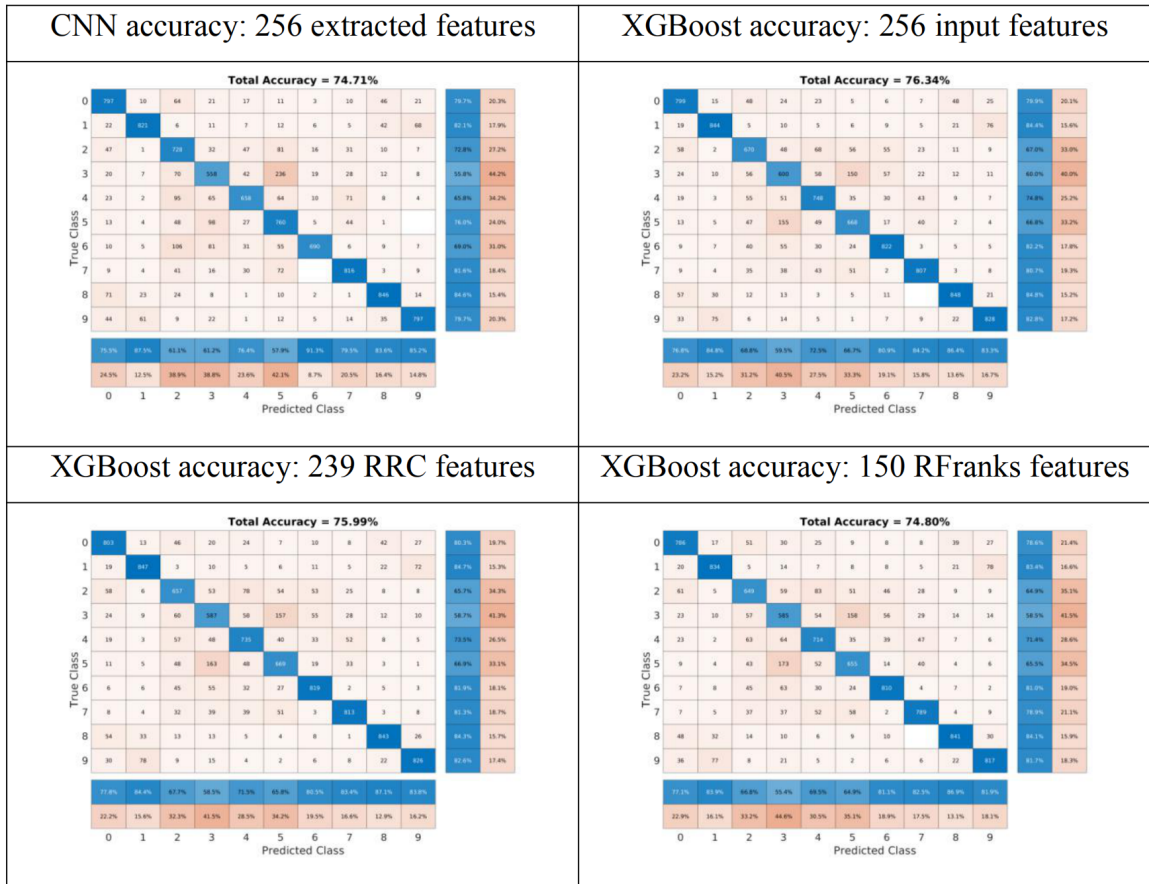


Figure A.1: Confusion matrices for AlexNet+XGBoost experiment.

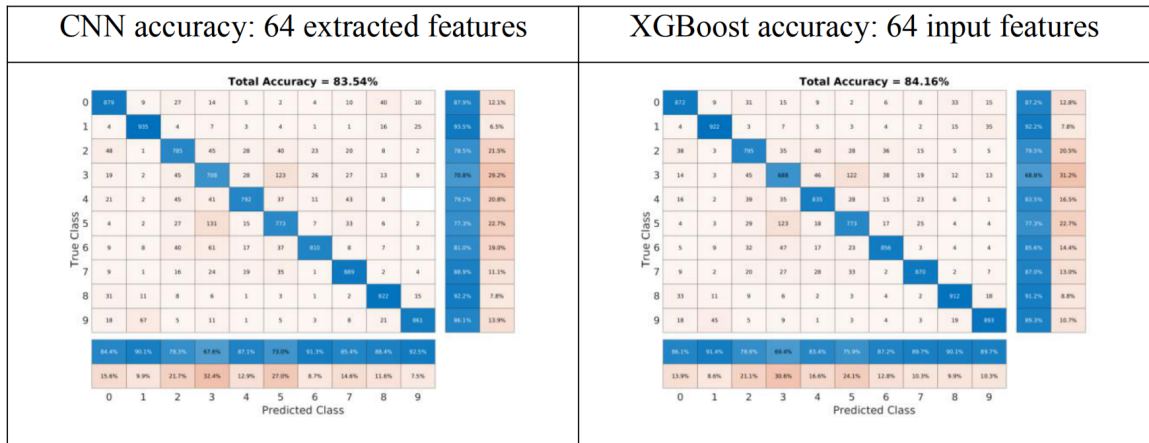


Figure A.2: Confusion matrices for AllConvNet + XGBoost experiment.

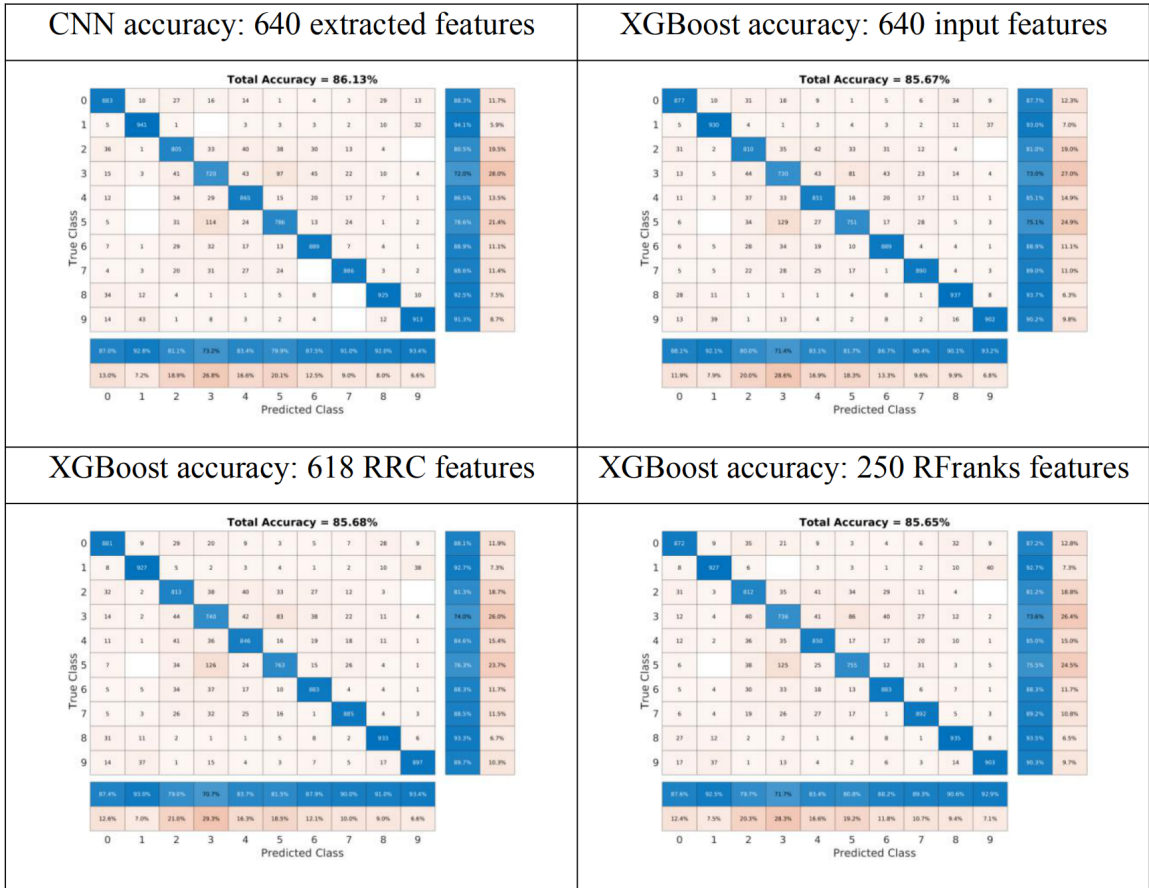


Figure A.3: Confusion matrices for WideResNet+XGBoost experiment.

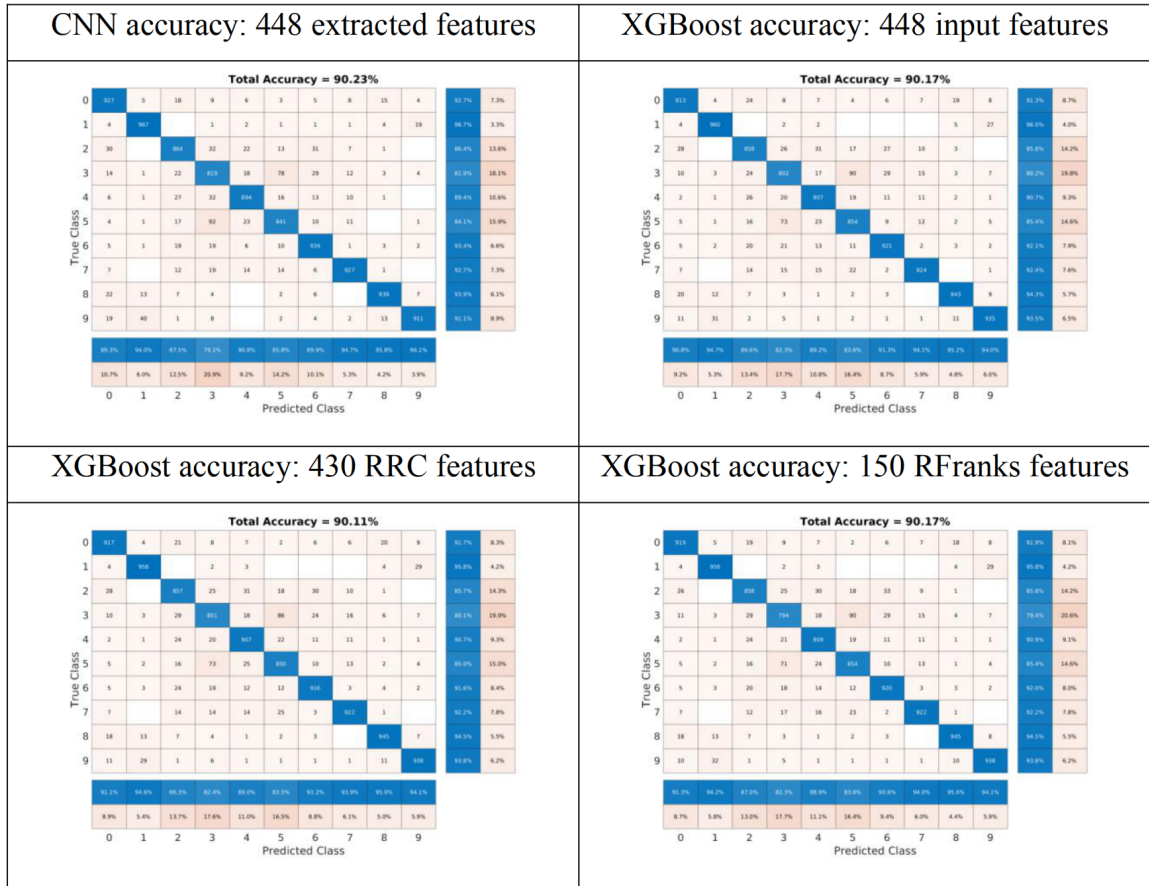


Figure A.4: Confusion matrices for DenseNet+XGBoost one-module experiment on original data.

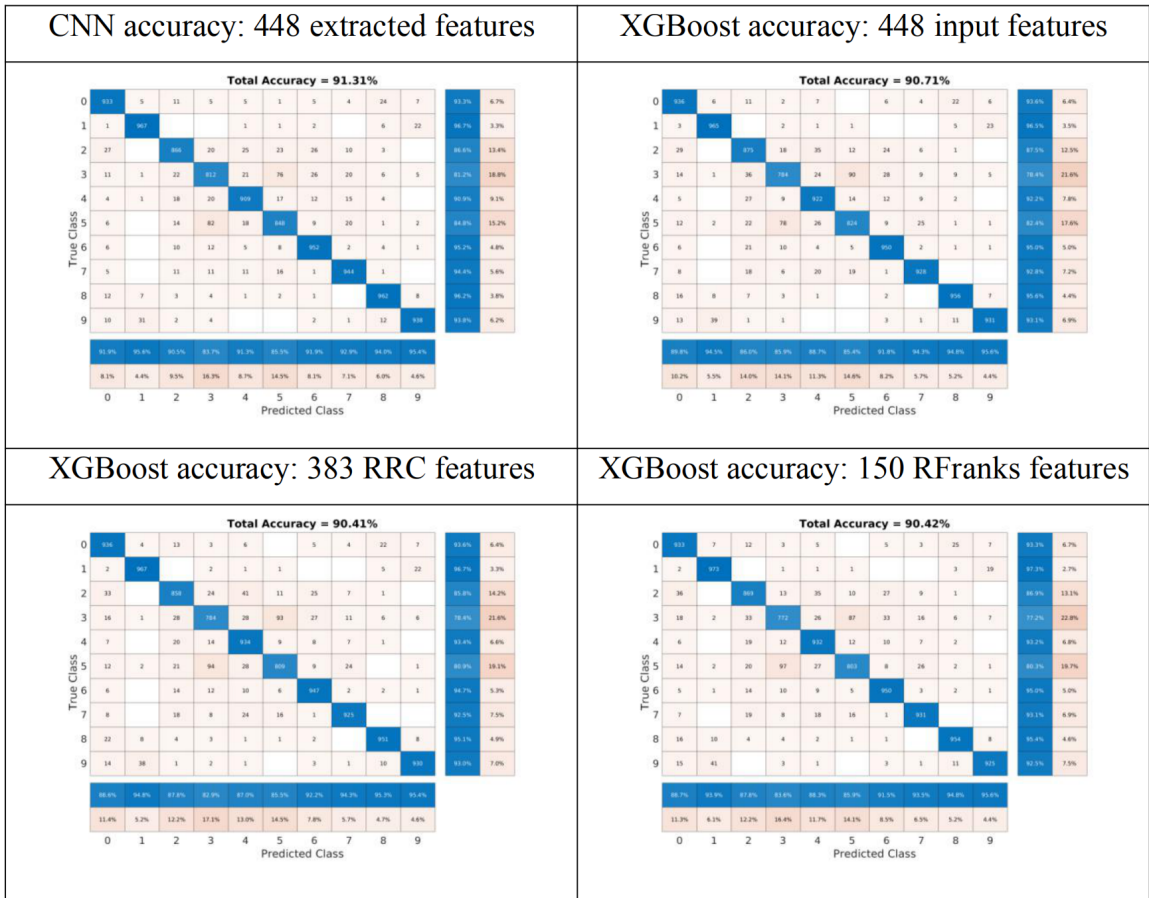


Figure A.5: Confusion matrices for DenseNet+XGBoost 5-fold CV experiment on original data.

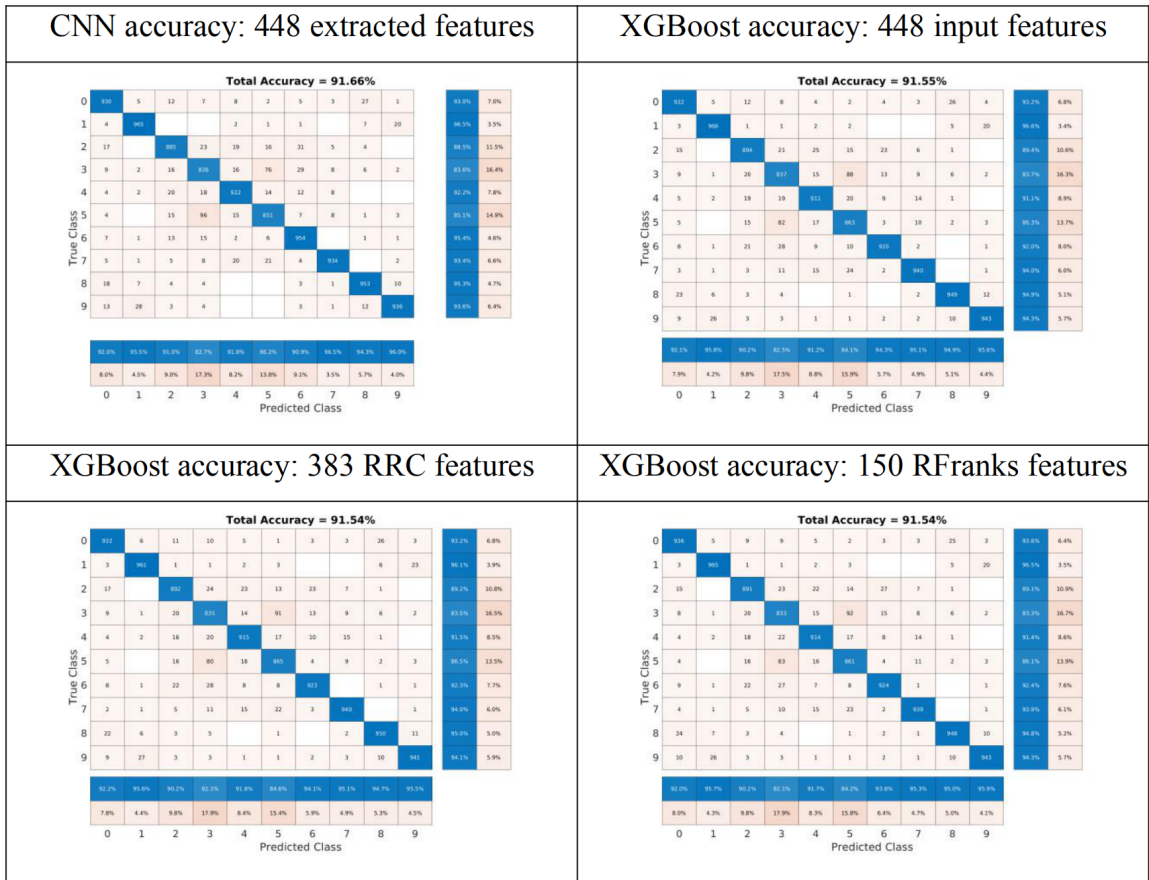


Figure A.6: Confusion matrices for DenseNet+XGBoost one-module experiment on the augmented data.

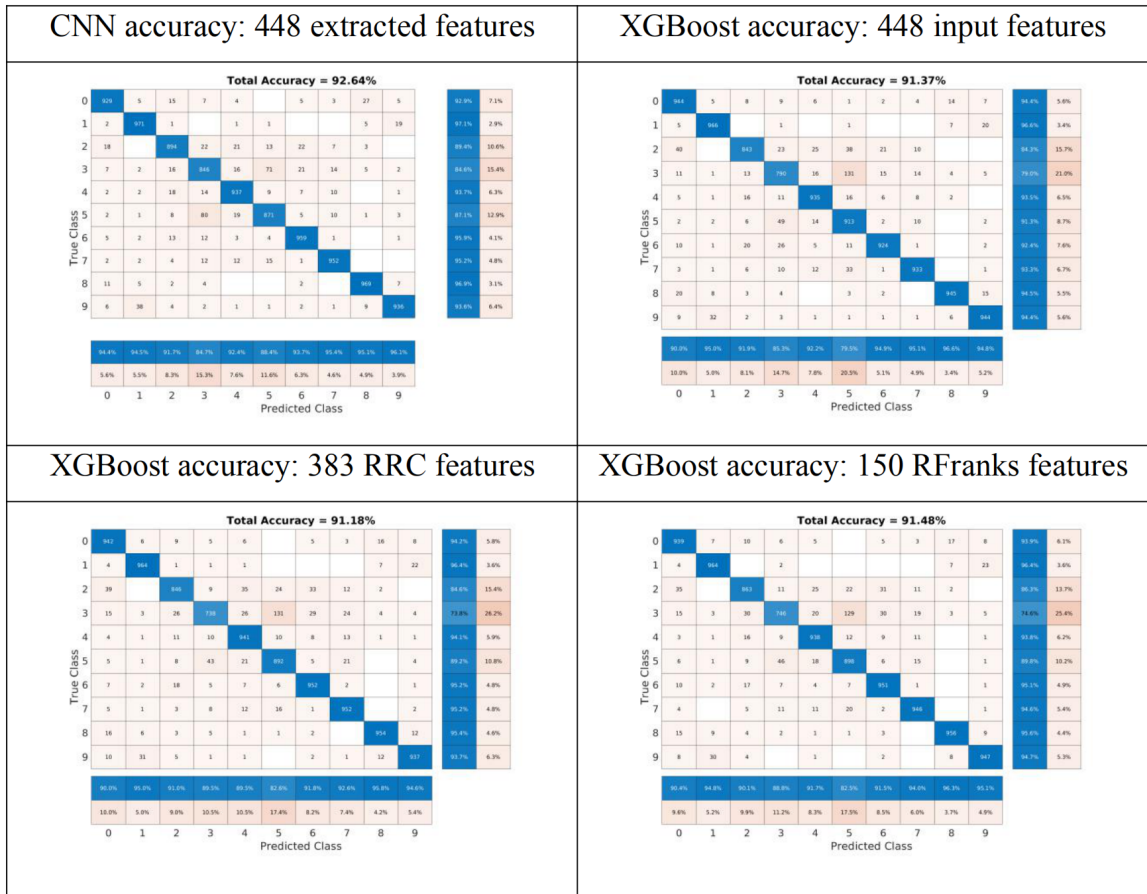


Figure A.7: Confusion matrices for DenseNet+XGBoost 5-fold CV experiment on the augmented data.

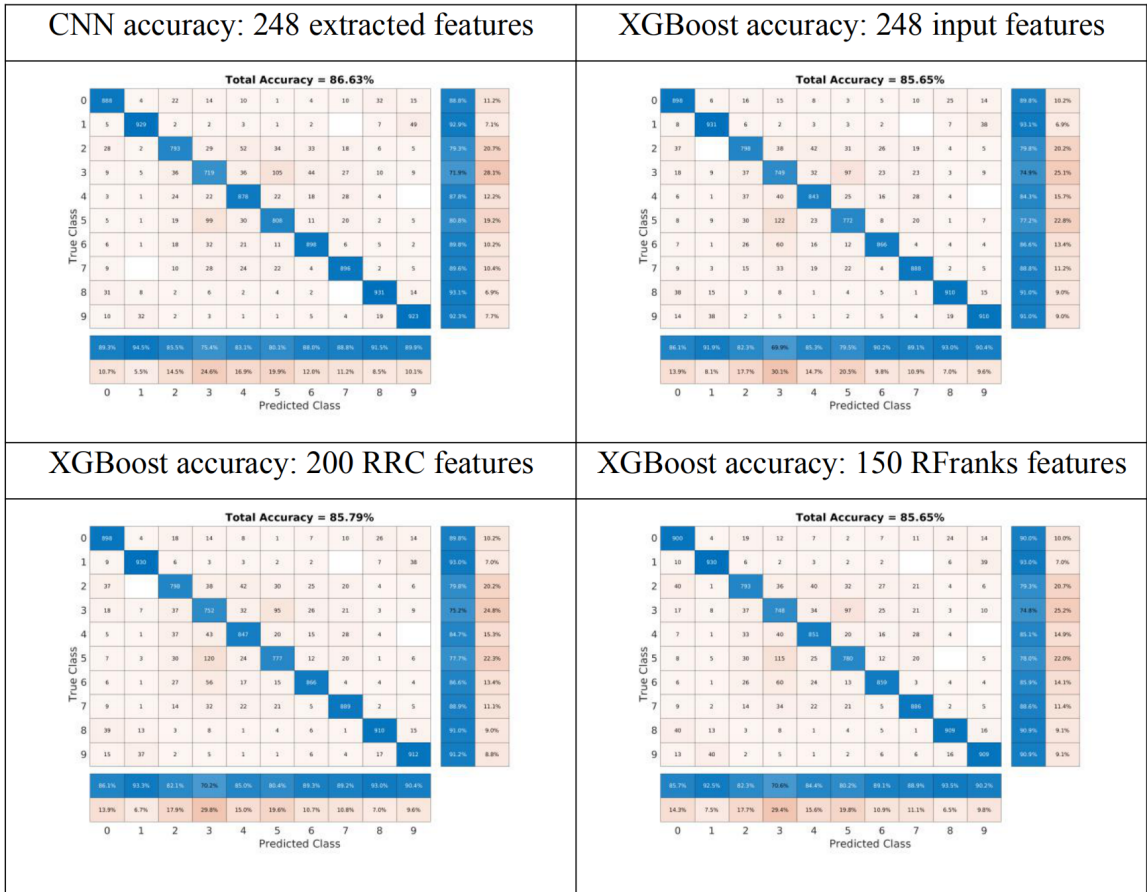


Figure A.8: Confusion matrices for CaffeNet+XGBoost transfer learning one-module experiment.

Bibliography

- Bekkerman, R. (2015). The present and the future of the kdd cup competition: an outsider’s perspective. <https://www.linkedin.com/pulse/present-future-kdd-cup-competition-outsiders-ron-bekkerman/> (accessed November 2020).
- Bennett, J. and S. Lanning (2007). The netflix prize. In *Proceedings of the KDD Cup Workshop 2007*, pp. 3–6.
- Bolon-Canedo, V., N. Sanchez-Marono, and A. Alonso-Betanzos (2013). A review of feature selection methods on synthetic data. *Knowledge and information systems* 34(3), 483–519.
- Boureau, Y., J. Ponce, and Y. LeCun (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 111–118.
- Burges, C. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23–581.
- Chandrashekar, G. and F. Sahin (2014). A survey on feature selection methods. *Computers and Electrical Engineering* 40(1), 16–28.

- Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. arXiv:1603.02754v3.
- Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss surfaces of multilayer networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Cover, T. M. (1974). The best two independent measurements are not the two best. *IEEE Transactions on Systems, Man, and Cybernetics 1*, 116–117.
- Freund, Y. and R. Schapire (1997). A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences 55*, 119–139.
- Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics 29*(5), 1189–1232.
- Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256.
- Goodfellow, I., Y. Bengio, and A. Courville (2016). Deep learning. *MIT Press*.
- Graham, B. (2015). Fractional max-pooling. arXiv:1412.6071.
- Gu, J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang, J. Cai, and T. Chen (2018). Recent advances in convolutional neural networks. *Pattern Recognition 77*, 354–377.

- Hastie, T., R. Tibshirani, and J. Friedman (2008). *The Elements of Statistical Learning*. Springer.
- He, X., J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela (2014). Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, ADKDD'14*.
- Hijazi, S., R. Kumar, and C. Rowen (2015). Using convolutional neural networks for image recognition. IP Group, Cadence.
- Hinton, G. E. and S. Roweis (2002). Stochastic neighbor embedding. *Advances in Neural Information Processing Systems, Cambridge, MA, USA, The MIT Press 15*, 833–840.
- Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). Improving neural networks by preventing co-adaptation of feature detectors. CoRR abs/1207.0580.
- Huang, G., Z. Liu, L. Van Der Maaten, and K. Weinberger (2016). Densely connected convolutional networks. arXiv:1608.06993.
- Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Journal of Machine Learning Research (JMLR)*, 448–456.
- James, G., D. Witten, T. Hastie, and R. Tibshirani (2017). An introduction to statistical learning with applications in r. *Springer Science+Business Media New York 440*.

- Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014). Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093.
- Kingma, D. and J. Ba (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.
- Krizhevsky, A., I. Sutskever, and G. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097–1105.
- LeCun, Y., Y. Bengio, and G. Hinton (2015). Deep learning. *Nature* 521, 436–444.
- LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11), 2278–2324.
- LeCun, Y. A., L. Bottou, G. B. Orr, and K.-R. Muller (2012). Efficient backprop. *Neural Networks: Tricks of the Trade - Second Edition*, 9–48.
- Lin, M., Q. Chen, and S. Yan (2014). Network in network. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Loshchilov, I. and F. Hutter (2017). Sgdr: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

- Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning (ICML), Volume 30*.
- McDonnell, M. and T. Vladusich (2015). Enhanced image classification with a fast-learning shallow convolutional neural network. arXiv:1503.04596.
- McNicholas, P. D. and P. Tait (2019). *Data Science with Julia*. Boca Raton: Chapman & Hall/CRC Press.
- Mishkin, D. and J. Matas (2016). All you need is a good init. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, pp. 807–814.
- Peng, H., F. Long, and C. Ding (2005). Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27(8), 1226–1238.
- Ponomareva, N., T. Colthurst, G. Hendry, S. Haykal, and S. Radpour (2017). Compact multi-class boosted trees. In *2017 IEEE International Conference on Big Data*, Boston, MA, pp. 47–56.
- Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks* 12(1), 145–151.
- Ren, X., H. Guo, S. Li, S. Wang, and J. Li (2017). A novel image classification method with cnn-xgboost model. *Lecture Notes in Computer Science* 10431, 378–390.

- Robust, P. L. (2010). Logitboost and adaptive base class (abc) logitboost. In *Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI'10)*, pp. 302–311.
- Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein (2015). Imagenet large scale visual recognition challenge. *International Journal of Conflict and Violence* 115(3), 211–252.
- Schaul, T., S. Zhang, and Y. LeCun (2013). No more pesky learning rates. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 343–351.
- Silver, D., J. Schrittwieser, and K. Simonyan (2017). Mastering the game of go without human knowledge. *Nature* 550, 354–359.
- Simonyan, K. and A. Zisserman (2015). Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*.
- Springenberg, J., A. Dosovitskiy, T. Brox, and M. Riedmiller (2014). Striving for simplicity: the all convolutional net. arXiv:1412.6806.
- Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1139–1147.
- Tang, J., S. Alelyani, and H. Liu (2014). Feature selection for classification: A review. *Data classification: algorithms and applications* 37.

- Van der Maaten, L. and G. Hinton (2008). Visualizing data using t-sne. *Journal of Machine Learning Research* 9, 2579–2605.
- Wan, A., L. Dunlap, D. Ho, J. Yin, S. Lee, H. Jin, S. Petryk, S. Bargal, and J. Gonzalez (2020). Nbdn: Neural-backed decision trees. arXiv:2004.00221.
- Wang, T., D. J. Wu, A. Coates, and A. Y. Ng (2012). End-to-end text recognition with convolutional neural networks. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pp. 3304–3308.
- Wijnhoven, R. G. J. and P. H. N. de With (2010). Fast training of object detection using stochastic gradient descent. In *International Conference on Pattern Recognition (ICPR)*, pp. 424–427.
- Zagoruyko, S. and N. Komodakis (2016). Wide residual networks. arXiv: 1605.07146.
- Zeiler, M. D. and R. Fergus (2014). Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pp. 818–833.
- Zeiler, M. D., M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, and J. Dean (2013). On rectified linear units for speech processing. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 3517–352.
- Zhang, S., A. E. Choromanska, and Y. LeCun (2015). Deep learning with elastic averaging sgd. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 685–693.

Zhao, Z., R. Anand, and M. Wang (2019). Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform. arXiv:1908.05376v.

Zinkevich, M., M. Weimer, L. Li, and A. J. Smola (2010). Parallelized stochastic gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 2595–2603.