Technische Universität München

DEPARTMENT OF MATHEMATICS

Residual Neural Network for Credit Scoring

Master Thesis

by

Ky Thoai Pham

Supervisor:	Prof. Dr. Claudia Czado
Advisor:	Prof. Dr. Claudia Czado
Submission Date:	19.09.2022

I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

Munich, 19.09.2022

Abstract

Nowadays, lending is one of the activities that contributes significantly to the income and profit of banks and fintech companies. Therefore, credit rating/scoring and risk assessment tools play very important roles to minimize credit risk, defined by the client's inability to repay the loan which the bank has granted. In this thesis, we will propose a novel method which utilizes residual neural networks for credit scoring. The proposed approach converts tabular data sets into black/white images and thus allows the application of a residual neural network with 50 layers (ResNet50) in credit scoring. Each pixel of the image corresponds to a feature bin of the tabular data set. The predictions from the ResNet50 are interpreted using the SHAP method. We did the experiments for the proposed method and logistic regression for two publicly available data sets of different sizes. Based on the results, our proposed approach shows superiority compared to logistic regression, especially, if the sample size is large.

Acknowledgement

I want to start by sincerely thanking Prof. Claudia Czado for giving me the opportunity to work on this topic under her supervision. I would like to thank her for all the guidance and fruitful discussions that significantly enhanced the progress of my work.
Secondly, I want to thank my quant team at the PwC Frankfurt. Thanks to their great support in my work, I have more time to focus on my thesis and have some new ideas for the model training part.
Lastly, I want to thank my family and friends, especially my mother and my father, for all the encouragement throughout my life. Your unconditional love and support in each step of the way is the foundation of my personal development.

Contents

Chapter 1

Introduction

The turmoil in the financial markets after the subprime mortgage crisis of 2007-2008 has emphasized that performance depends on consumer lending. Nowadays, the majority of a bank's profits come from lending activities. Credit granting is one of the activities that contribute significantly to the income and profit of banks, but it also entails numerous possible risks (Zakrzewska [2007]). The main risk of this activity for the bank is the client's inability to repay the loan that the bank has granted. Such clients are called default/bad clients. On the other hand, the decision of whether to provide a loan to a client often depends on the qualifications and experience of the credit appraisers (Thomas [2000]). In addition, the basis for granting credit to a client is based on a number of rating criteria, some of which are very difficult to measure accurately. For example, the **5C** (Abrahams and Zhang [2008]) standard when granting credit is based on the bank's assessment of the applicant's qualifications, capacity, capital, collateral, and conditions. It goes without saying that some criteria, such as the applicant's qualifications, are difficult factors to assess and thus can lead to mistakes in credit-granting decisions.

In addition, the **5C**-based credit rating method is expensive, and there can be inconsistencies in credit-granting decisions between different credit appraisers for the same application. Because of these limitations, banks as well as financial institutions need to use reliable, consistent, objective, and low-cost methods for credit evaluation (Akhavein, Frame and White [2005]). In addition, according to Thomas [2002], banks need a credit rating method that satisfies the following criteria: (1) low cost and easy to operate, (2) fast and stable, (3) consistent decision based on the objective information that is independent of human emotions and subjective feelings, and (4) the effectiveness of the credit rating methodology can be easily checked and adjusted at any time in order to timely adjust to changes in policy or economic conditions.

For the credit scoring problem to discriminate default from non-default clients, the traditional approach is based on statistical methods such as multivariable linear regression (Meyer and Pifer [1970]), and logistic regression (Desai, Jonathan, and George [1996]). In recent

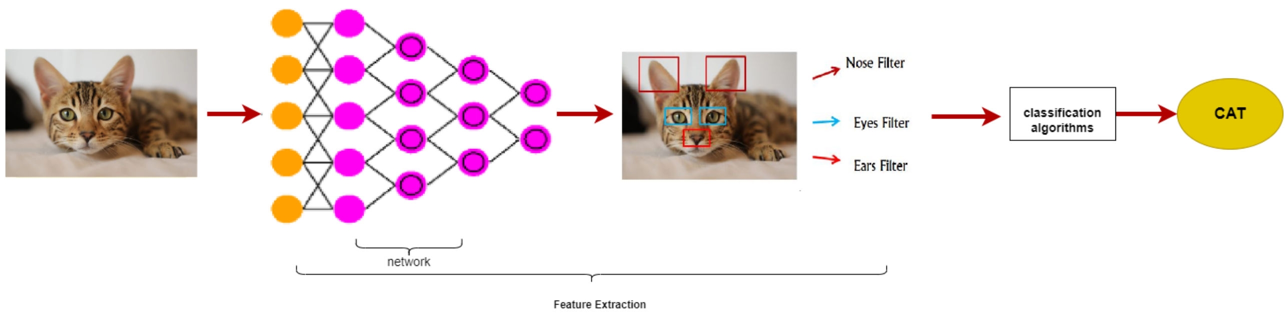


Figure 1.1: An example of image classification for the cat classification.

years, there have appeared a number of new credit classification models utilizing approaches of machine learning (Provenzano et al. [2020]) and artificial intelligence (Liu, Huang and Lu [2019]). Unlike previous approaches, these new methods do not make any of the strict model assumptions required by statistical approaches. Instead, these new approaches try to seek to exploit and generate insights to produce output or predictions based only on observations and historical data. Some typical machine learning models such as artificial intelligence network (Huang, Chen, Hsu, Chen and Wu [2004]), support vector machine (Huang, Chen, Hsu, Chen and Wu [2004]), k-nearest neighbors (Aida [2015]) and random forest (Sharma [2011]) have demonstrated many advantages in terms of accuracy and reliability compared to some traditional classification models.

Following this trend, in this thesis, we will propose a method which applies the knowledge of **computer vision** and **convolutional neural network** (LeCun [1998]) to solve the credit scoring problem and study its performance on the two publicly available data sets: **German Credit** (Hans [1995]) and **Home Credit** (Kaggle [2018]). The main idea of our method is based on **image classification**, one of the most fundamental tasks in computer vision. It is the task of assigning one (single-label classification) or more (multi-label classification) labels to a given image by discovering patterns and characteristics in the image. Figure 1.1 represents an example of a cat image which is correctly classified by detecting characteristics of the nose, eyes and ears via a neural network. We apply this idea to classify default or non-default by proposing a way to convert the tabular information of a client, such as age, salary and health, to an image format. Next, we apply a convolutional neural network to predict default or non-default.

This thesis is organized as follows. Chapter 2 gives a review of the theoretical concepts of (1) methods for transforming tabular data to the image format, (2) convolutional neural networks and residual neural networks with 50 layers (ResNet50) and (3) the SHAP method for model interpretation. Chapter 3 contains the model performances of the ResNet50 and logistic regression for the two above-mentioned data sets and model interpretation for the ResNet50. Chapter 4 gives the conclusions and outlines the future work.

Chapter 2

Theoretical background

2.1 Feature transformation to the image format

2.1.1 Weight of evidence (WOE) and information value (IV)

WOE and IV (Siddiqi [2005]) are common methods for feature transformation and selection. They are widely used in credit risk models to measure the power to classify good and bad applicants separately and to transform the continuous and categorical independent variables into discrete categories automatically. Moreover, the advantages of the WOE are:

- Tackle missing/not available information and outliers.
- No need for performing dummy encoding.
- By applying a proper binning technique, monotonic relationships between the independent and the dependent variables can be detected. For example, the higher an independent variable is, the higher the probability of default is.
- Evaluate contribution of the independent variables to the target by applying the IV.

Background

Suppose that we have a binary dependent random variable Y and a set of independent variables $X_1, \dots, X_p$ with observations $x_1, \dots, x_p$.
The WOE/IV method is based on the following relationship for the j-th independent variable:

$$\log \frac{P(Y=1|x_j)}{P(Y=0|x_j)} = \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{unconditional\ log\text{–}odds} + \underbrace{\log \frac{f_j(x_j|Y=1)}{f_j(x_j|Y=0)}}_{WOE(x_j)}, \tag{1}$$

where $f_j(x_j|y)$ denotes the conditional probability density function at x_j of the covariate X_j given $Y = y$. This relationship expresses that the conditional logit of $P(Y = 1|x_j)$ can be written as the sum of the overall log-odds (i.e., the "intercept") and the log-density ratio (also known as the WOE).

It is evident that the WOE and the conditional log odds of $Y = 1$ are perfectly dependent, as the "intercept" is constant. Hence, the higher the value of the WOE is, the higher the probability of observing $Y = 1$ is. In fact, when the WOE is positive, the probability of observing $Y = 1$ is higher than average and vice versa when the WOE is negative. The log odds are simply equal to the unconditional log odds when the WOE is equal to 0. We see that the left-hand side of the equation (1) is exactly what a logistic regression model should predict. Hence, when training a logistic regression model, the WOE is hence likely the most popular method for feature transformation in the credit scoring industry.

We can use the WOE to measure the predictive power of x_j – i.e., how well it helps us to classify goods ($Y = 0$) and bads ($Y = 1$) correctly. This is estimated by the information value (IV) for the variable X_j which is defined as:

$$IV_j = \int \log(\frac{f_j(x_j|Y = 1)}{f_j(x_j|Y = 0)})(f_j(x_j|Y = 1) - f_j(x_j|Y = 0))dx_j. \tag{2}$$

The IV is a weighted "sum" of all the individual WOE values where the weights include the absolute difference between the numerator and the denominator (the WOE captures the relative difference). While $(f_j(x_j|Y = 1) - f_j(x_j|Y = 0))$ can be negative, the integrand in (2) is always positive.

Estimating the WOE

The most common approach to estimate the conditional densities needed to calculate the WOE is to select bins for the variable X_j and then use a histogram-type estimation.
If $B_1, \dots, B_{K_j}$ denote the bins for X_j, the WOE of X_j for bin k can be estimated as:

$$\widehat{WOE}_k = \log\frac{\hat{P}(X_j \in B_k|Y = 1)}{\hat{P}(X_j \in B_k|Y = 0)}, \text{for } k = 1, ..., K_j$$

where $\hat{P}(X_j \in B_k|Y = 1)$ and $\hat{P}(X_j \in B_k|Y = 0)$ are estimated as:

$$\hat{P}(X_j \in B_k|Y = 1) = \frac{n_{Y=1,X_j \in B_k}}{n_{Y=1,X_j \in B_k} + n_{Y=0,X_j \in B_k}}$$

$$\hat{P}(X_j \in B_k|Y = 0) = \frac{n_{Y=0,X_j \in B_k}}{n_{Y=1,X_j \in B_k} + n_{Y=0,X_j \in B_k}}$$

where $n_{Y=1,X_j \in B_k}$ is the number of observations $i \in \{1, .., n\}$ with $Y_i = 1$ and $x_{ij} \in B_k$ and

$n_{Y=0,X_j \in B_k}$ is the number of observations $i \in \{1,..,n\}$ with $Y_i = 0$ and $x_{ij} \in B_k$, respectively. This means that the IV for variable X_j can be estimated as:

$$\widehat{IV}_j = \sum_{k=1}^{K_j} (\hat{P}(X_j \in B_k | Y = 1) - \hat{P}(X_j \in B_k | Y = 0)) \times \widehat{WOE}_k.$$

According to Siddiqi [2005], by convention the value of the IV statistic can be interpreted as follows. If the IV statistic is:

- Less than 0.02, then the predictor has no relationship to the goods versus bads odds ratio
- 0.02 to 0.1, then the predictor has only a weak relationship to the goods versus bads odds ratio
- 0.1 to 0.3, then the predictor has a medium strength relationship to the goods versus bads odds ratio
- 0.3 or higher, then the predictor has a strong relationship to the goods versus bads odds ratio.

2.1.2 Algorithm for transforming the data into the WOE vectors

Classification and regression trees (CART)

Classification and regression trees (CART) (Breiman [1984]) is a classification method used to construct decision trees by using the Gini's impurity as a criterion for splitting. This algorithm works by splitting a node into two child nodes repeatedly. Its steps will be presented as below:

- Step 1: For each independent variable X_j with the number of unique values D_j, we find the best split by considering $D_j - 1$ possible splits for each independent variable X_j with D_j different values. We take the split which maximizes the splitting criterion. The resulting set of splits contains the best splits (one for each independent variable).
- Step 2: We find the node's the best split by taking the split of the variable X_j^* which maximizes the criterion among the best splits from Step 1.
- Step 3: Using the optimal node split from Step 2 to split the node and repeat from Step 1 until the stopping criterion is satisfied. The stopping criterion is defined by the user. For example, it can be that the maximum depth of a tree is less than a pre-specified number or the number of observations in the node is less than a pre-specified limit.

For a binary classification, we use the Gini's impurity index for the splitting criterion. The Gini's impurity index is estimated for node t as:

$$\hat{G}(t) = 1 - \hat{P}(1|t)^2 - \hat{P}(0|t)^2,$$

where $\hat{P}(1|t) = \frac{\#\{i\colon x_{ij} \in \text{ node } t, Y_i=1\}}{\#\{i\colon x_{ij} \in \text{ node } t\}}$ and $\hat{P}(0|t) = \frac{\#\{i\colon x_{ij} \in \text{ node } t, Y_i=0\}}{\#\{i\colon x_{ij} \in \text{ node } t\}}$

For example, we have note $t : X_j \leq$ c, we define

$$\begin{aligned} n_t &= \#\{i : x_{ij} \leq c\} \\ n_{good,t} &= \#\{i : x_{ij} \leq c, Y_i = 0\} \\ n_{bad,t} &= \#\{i : x_{ij} \leq c, Y_i = 1\} \end{aligned}$$

then

$$\hat{G}(t) = 1 - \hat{P}(bad|t)^2 - \hat{P}(good|t)^2 = 1 - (\frac{n_{bad,t}}{n_t})^2 - (\frac{n_{good,t}}{n_t})^2.$$

Chi-square automatic interaction detection (CHAID)

Chi-squared automatic interaction detection (CHAID) (Kass [1980]), is a classification algorithm for constructing decision trees by using the χ^2 statistics to find the best split. Only categorical independent variables are accepted by the CHAID. Continuous independent variables are converted into ordinal variables before usage.

Merging The main idea of this step is that, for each independent variable X_j, we merge categories which are most similar to each other. Each final category of the independent variable will lead to one child node if the independent variable X_j is used to split the node. The merging step also computes the adjusted p-value and this will be used in the splitting step. In more detail, the following steps need to be performed:

1. If the number of categories for the independent variable X_j is 1, stop and the adjusted p-value will be set to be 1.

2. If the number of categories for the independent variable X_j is 2, go to Step 8.

3. Else, for all categories of independent variable X_j, find pairs of categories which are most similar. The most similar pair is defined as the pair whose test statistic has the highest value for p-value with respect to the dependent variable Y. The formulation of the test statistic and its p-value estimation will be described later.

4. For the pair with the largest p-value (most similar), if its p-value is greater than α_{merge}, which is specified by the user, this pair is merged into a single compound category. Then

a set of new categories of the independent variable X_j is created. If p-value is less than α_{merge}, then go to Step 7.

5. $(Optional)$ If three or more original categories are included in the new set, then we find the best binary split within the compound category for which the p-value is the smallest. Perform this binary split if its p-value is not greater than an $\alpha_{split-merge}$, which is specified by the user.

6. Go to Step 2.

7. $(Optional)$ Any category having too few observations (as compared with the user-specified minimum segment size) is merged with the most similar other category, as determined by the largest of the p-value.

8. The adjusted p-value is computed for the merged categories by applying the Bonferroni adjustments.

P-value calculation Calculations of p-value depend on the type of dependent variable. The merging step needs the p-value for a pair of categories of X_j, and sometimes needs the p-value for all categories of X_j. When p-value for a pair of X_j categories is needed, only part of data in the current node is relevant. Let $\mathcal{D}$ denotes the relevant data. Suppose in $\mathcal{D}$, X_j takes on values $\{x_{j1}, ..., x_{jD_j}\}$ and there are 2 categories of Y (binary). Since the dependent variable Y is binary, the null hypothesis of independence of the variable X_j and the dependent variable Y is tested. To do the test, a contingency table is formed using classes of Y as columns and categories of the predictor X_j as rows. The expected cell frequencies under the null hypothesis are estimated. The observed cell frequencies and the expected cell frequencies are used to calculate the Pearson χ^2 statistic (Pearson [1900]) or the likelihood ratio statistic (King [1989]). The p-value is computed based on either one of these two statistics. Under the null hypothesis of independence of Y and X_j, the Pearson's χ^2 statistic or the likelihood ratio statistic are respectively,

$$\chi^2 = \sum_{c=1}^{2}\sum_{d=1}^{D_j} \frac{(n_{dc} - \hat{m}_{dc})^2}{\hat{m}_{dc}}$$

$$LR^2 = 2\sum_{c=1}^{2}\sum_{d=1}^{D_j} n_{dc} \ln(\frac{n_{dc}}{\hat{m}_{dc}}),$$

where n_{dc} is the observed cell frequency and $\hat{m}_{dc}$ is the estimated expected cell frequency for the cell $(x_{ij} = x_{jd}, Y_i = c)$. Their formulations are given as follows:

$$n_{dc} = \sum_{i=1}^{n} 1_{\{x_{ij}=x_{jd}, Y_i=c\}}$$

$$m_{dc} = \frac{n_{d.}n_{c.}}{n_{..}}$$

with

$$n_{d.} = \sum_{c=1}^{2} n_{dc},\ n_{c.} = \sum_{d=1}^{D_j} n_{dc},\ n_{..} = \sum_{c=1}^{2}\sum_{d=1}^{D_j} n_{dc}.$$

Then, the corresponding p-value is given by:

- $p = P(\chi_d^2 > \chi^2)$ for the Pearson's χ^2 test
- $p = P(\chi_d^2 > LR^2)$ for the likelihood ratio test

where χ_d^2 follows a χ^2 distribution with degrees of freedom $d = (J-1)(I-1)$.

Bonferroni adjustments The adjusted p-value is calculated as the p-value times a Bonferroni multiplier. The Bonferroni multiplier adjusts for multiple tests. Consider an independent variable with I original categories that, after the merging step, is reduced to r categories. The number of ways that I categories can be merged into r categories is known as the Bonferroni multiplier B. For $r = I, B = 1$. For $2 \le r \le I$, use the equation derived by Kass [1980]

$$B = \sum_{v=0}^{r-1} (-1)^v \frac{(r-v)^I}{v!(r-v)!}.$$

Splitting The "best" split for each independent variable is found in the merging step. The splitting step selects which independent variable to be used to best split the node. The selection is accomplished by comparing the adjusted p-value associated with each independent variable. The adjusted p-value is obtained as the merging step. Its steps will be presented as below:

1. Select the independent variable X_j^* that has the smallest adjusted p-value (i.e., most significant for the dependence of Y and the independent variable X_j).
2. Using the "best" split of the variable from Step 1 to split the node and repeat from the merging step and Step 1 until the stopping criterion is satisfied. The stopping criterion is defined by the user, for example, the maximum depth of a tree is less than a pre-specified number or the number of observations in the node is less than a pre-specified limit.

For example, we have the dependent variable Y and two independent variables X_1 and X_2. Y is the binary variable, X_1 is the categorical one with 5 values: 1, 2, 3, 4 and 5 and X_2 is the categorical one with 3 values: 1, 2 and 3. Next, we perform the merging step on X_1 and X_2. Based on the results of the merging step, suppose that X_1 has the smallest adjusted p-value (Step 1) and we use the "best" split of X_1 from the merging step to split the node (Step 2). Therefore, for example, we have node $t_1 = \{i : x_{i1} \in \{1, 2, 3\}\} = n_{t_1}$ and $t_2 = \{i : x_{i1} \in \{4, 5\}\} = n_{t_2}$. Next, we repeat the merging step, Step 1 and Step 2, on n_{t_1} and n_{t_2} respectively, to split these nodes. We repeat these 3 steps on the next nodes until the stopping criterion is satisfied.

Algorithm for transforming the data into the WOE vectors

The algorithm for searching for the best encoding of predictors is as follows:

- For continuous independent variables, first a default encoding is derived using the CART. If the number of categories is fewer than 20, the algorithm will explicitly search through all possible partitions (combinations of default groups) to achieve the least number of groups with the greatest estimated the information value ($\widehat{IV}$). When the number of categories is greater than 20, the CHAID will be used.
- For categorical (discrete) predictors, the default (original) grouping is adjusted by using the CHAID algorithm with the differences in the WOE between bins as the criterion for combining/splitting.

2.1.3 Conversion of tabular data into images

After each variable is transformed into the categorical format of the WOE bins, all predictors have the categorical format now. Next, we will transform the categorical format of the predictors of each applicant to an image. Let's say that we have an applicant i with p predictors, $\mathbf{x}_i = (x_{i1}^{woe}, ..., x_{ip}^{woe})^T$, where x_{ij}^{woe} is the bin number $\in \{1, ..., K_j\}$ of predictor X_j for $j \in \{1, ..., p\}$ for client i. Then, the feature image representation I_i of this record is

$$\mathbf{I}_i = [\mathbf{f}_1(x_{i1}^{woe}), \mathbf{f}_2(x_{i2}^{woe}), ..., \mathbf{f}_p(x_{ip}^{woe})], \tag{3}$$

where $\mathbf{f}_j$: $x_j \mapsto \{0, 1\}^K$ is a function to perform dummy encoding for a predictor x_j and K is defined as a maximum number of distinct bins of all features, i.e $K = \max\{K_j\}_{j=1}^p$.

age	salary (Euro)	job_type
[-Inf,18)	[-Inf,40000)	full-time
[18,50)	[40000,80000)	part-time
[50,Inf)	[80000,Inf)	

Table 2.1: Table of the WOE bins for example.

For example, we have a data set with three predictors which are age (numeric), salary (numeric) and job type (qualitative). By applying the WOE transformation, the optimal bins for

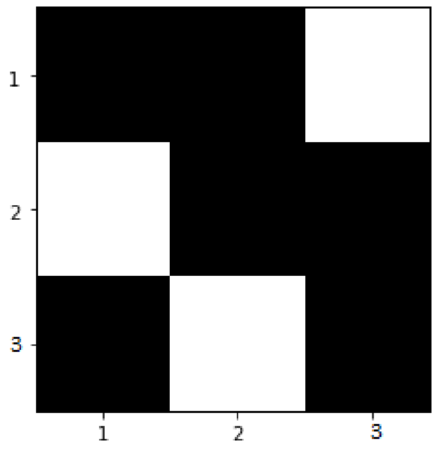


Figure 2.1: Feature image representation of the applicant A whose age is [18,50) (bin 2 - attribute 1), salary is in [80000, Inf) (bin 3 - attribute 2) and job type is full-time (bin 1 - attribute 3).

these 3 variables are given in Table 2.1.
Let's say that an applicant A whose age is 40 has a full-time job with a salary of 85000 Euro. By applying Table 2.1, we have

$$\mathbf{x}_A = (x^{woe}_{A,age}, x^{woe}_{A,salary}, x^{woe}_{A,job\_type})^T = ([18,50),[40000,80000),\text{full-time})^T.$$

Next, by applying equation (3) with $K = \max\{K_{age}, K_{salary}, K_{job\_type}\} = \max\{3,3,2\} = 3$, we have

$$\mathbf{f}(x^{woe}_{age}) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\mathbf{f}(x^{woe}_{salary}) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\mathbf{f}(x^{woe}_{job\_type}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

This implies the matrix $\mathbf{I}_A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ which is represented as the binary image given in Figure 2.1.

2.2 Neural networks

An artificial neural network is a non-linear application in terms of its input parameters $\boldsymbol{\theta}$ that associate to an entry $\mathbf{x}$ an output $y = f(\mathbf{x}, \boldsymbol{\theta})$. Y is assumed to be unidimensional for simplicity, but it could actually be multidimensional. For this application f has a particular form that we will make. The neural networks can be applied to classification or regression. The parameters $\boldsymbol{\theta}$ are estimated from a learning sample, as is customary in machine learning. The function to minimize is not convex, leading to local minimizers. The method's success was based on a universal approximation theorem because of Cybenko [1989] and Hornik [1991]. Furthermore, the backpropagation of the gradient, which LeCun [1986] proposed as an effective method of computing a neural network's gradient, makes it simple to find a local minimizer of the quadratic criterion.

2.2.1 Artificial neuron

An artificial j-th neuron of a network is a function f_j of the input $\mathbf{x} = (x_1, \ldots, x_p)^T$ weighted by a vector of connection weights $\mathbf{w}_j = (w_{j,1}, ..., w_{j,p})^T$, completed by a neuron bias $b_j \in \mathbb{R}$, and associated to an activation function $\phi : \mathbb{R} \to \mathbb{R}$, given by

$$\begin{aligned} h_j = f_j(\mathbf{x}) &= \phi(\mathbf{w}_j^T \mathbf{x} + b_j) \\ &= \phi(\sum_{k=1}^{p} w_{jk} x_k + b_j). \end{aligned}$$

Some activation functions are usually considered.

$$\begin{aligned} &\text{The identity function } \phi(x) = x,\ x \in \mathbb{R} \\ &\text{The sigmoid function } \phi_s(x) = \frac{1}{1+\exp(-x)} \qquad (4) \\ &\text{The hyperbolic tangent function ("tanh") } \phi_t(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{\exp(2x) - 1}{\exp(2x) + 1} \\ &\text{The hard threshold function } \phi_\beta(x) = \mathbb{1}_{x \geq \beta} \\ &\text{The rectified linear unit (ReLU) activation function } \phi_{ReLu}(x) = \max(0, x) \end{aligned}$$

The Figure 2.2 represents a sample artificial neuron, and Figure 2.3 represents the activation functions described above.

2.2.2 Multilayer perceptron

A neural network is made up of vertically stacked components, called **layers**. Usually, a neural network has 3 types of layers.

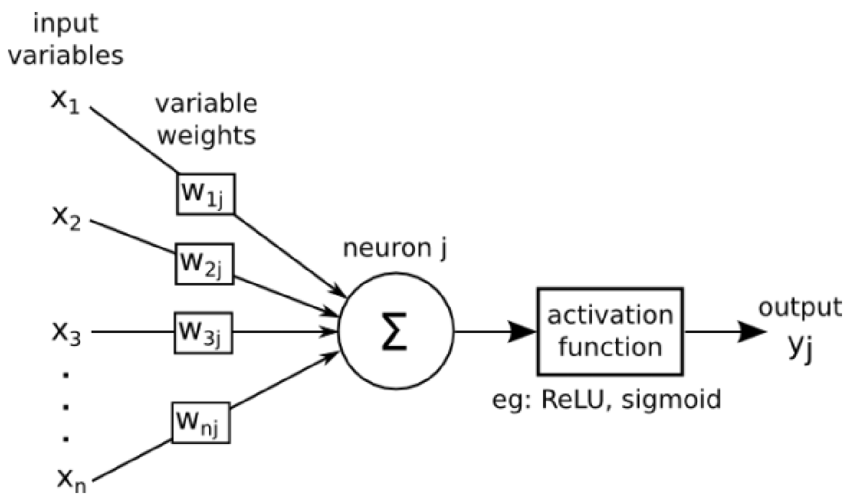


Figure 2.2: A sample artificial neuron. (Source: andrewjames turner.co.uk).

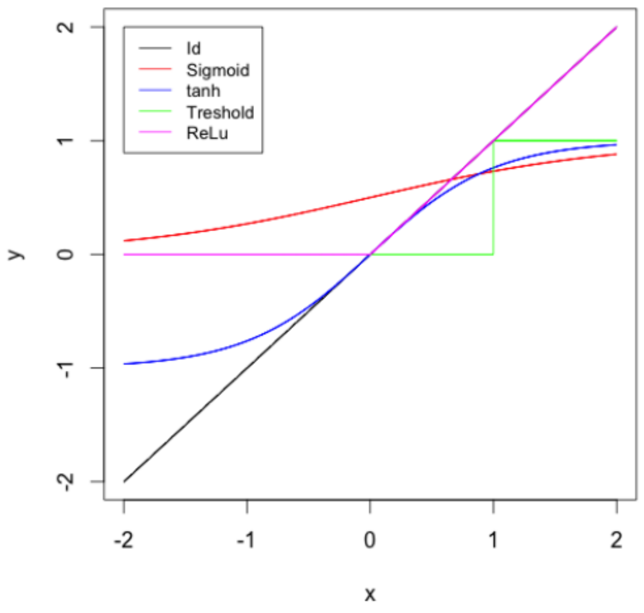


Figure 2.3: Activation functions.

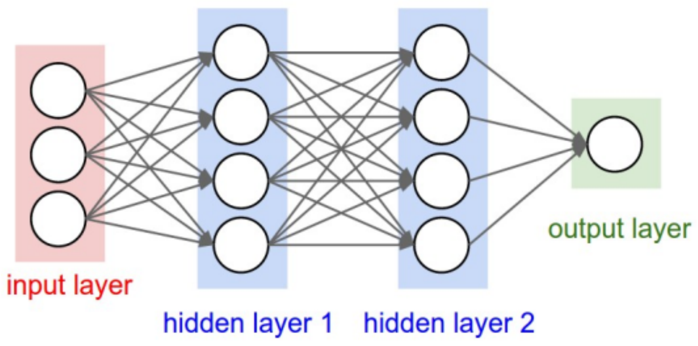


Figure 2.4: A basic neural network. (Source : http://blog.christianperone.com).

- Input layer – the first type is the **input layer**. This layer accepts the data as input and pass it to the next layers.
- Hidden layer – the second type is the **hidden layer**. A neural network contains one or more hidden layers. Hidden layers are responsible for the "learning" mechanism of neural networks. They perform multiple functions at the same time such as data transformation, feature creation, etc.
- Output layer – the last type of layer is the **output layer**. The output layer holds the result or the output of the problem.

A **multilayer perceptron** (or **neural network**) is a structure composed by one input layer, one output layer and several hidden layers of neurons. Regarding these hidden layers, the output of the first hidden layer will become the input of the next hidden layer. On the last layer, we may apply a different activation function to the output of hidden layers depending on the type of problem which is regression or classification. The Figure 2.4 represents a neural network with three input variables, one output variable, and two hidden layers (each of the hidden layers has 4 neurons).

A multilayer perceptron is an architecture where each neuron of a layer is connected to all the units of the next layer, but has no link with the neurons of the same layer. The **parameters** of the network architecture are the number of **hidden layers** and of **neurons** in each layer. The activation functions are also chosen by the user. As was previously mentioned, the activation function for the output layer is typically different from that of the hidden layers. In the case of regression, we use no activation function for the output layer. For binary classification, the output gives a prediction of $P(Y = 1|\mathbf{X})$. Since this value is in $[0, 1]$, the sigmoid activation function (given in (4)) is generally utilized. For multiclass classification, the output layer contains one neuron per class i, giving a prediction of $P(Y = i|\mathbf{X})$. The sum of all these values

has to be equal to 1. The multidimensional function softmax is generally used. However, in this thesis, we only focus on binary classification.
Let's summarize the mathematical formulation of a multilayer perceptron with L hidden layers. We set $\mathbf{h}^0 = \mathbf{x} \in \mathbb{R}^p$.

- For $k = 1, ..., L$ (hidden layers), assuming there are N_k neurons in the hidden layer k, we have

$$\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$$
$$\mathbf{h}^{(k)} = \boldsymbol{\phi}(\mathbf{a}^{(k)}) := \begin{pmatrix} \phi(a_1^{(k)}) \\ \vdots \\ \phi(a_{N_k}^{(k)}) \end{pmatrix}$$

  with $\boldsymbol{W}^{(k)} \in \mathbb{R}^{N_k \times p}$ for $k = 1$, $\boldsymbol{W}^{(k)} \in \mathbb{R}^{N_k \times N_{k-1}}$ for $k = 2, ..., L$ and $\boldsymbol{b}^{(k)} \in \mathbb{R}^{N_k}$.

- For $k = L + 1$ (output layer), we have

$$a^{(L+1)} = b^{(L+1)} + \mathbf{W}^{(L+1)}\mathbf{h}^{(L)}$$
$$h^{(L+1)} = \psi(a^{(L+1)}) := \hat{f}(\mathbf{x}, \boldsymbol{\theta})$$

  with $\boldsymbol{W}^{(L+1)} \in \mathbb{R}^{1 \times N_L}$ and $b^{(L+1)} \in \mathbb{R}$.

where ϕ is the activation function for $k = 1, ..., L$ hidden layers and ψ is the output layer activation function. For binary classification problem, ψ is a sigmoid function given in (4). The $\mathbf{h}^{(k)}$, $\mathbf{b}^{(k)}$ and $\mathbf{W}^{(k)}$ is the neuron, bias and a weight matrix respectively at layer k.

2.2.3 Universal approximation theorem

Hornik [1991] showed that any bounded and regular function $f : \mathbb{R}^p \to \mathbb{R}$ can be approximated at any given precision by a neural network with one hidden layer containing a finite number of neurons, having the same activation function, and one linear output neuron. This result was earlier proved by Cybenko [1989] in the particular case of the sigmoid activation function. More precisely, the Hornik's theorem can be stated as follows.

THEOREM 1. *Let ϕ be a bounded, continuous and non-decreasing (activation) function on $\mathbb{R}$. Let K_p be some compact set in $\mathbb{R}^p$ and $\mathcal{C}(K_p))$ the set of real valued continuous functions defined on K_p. Let $f \in \mathcal{C}(K_p))$. Then for all $\epsilon > 0$, there exists $N \in \mathbb{N}$, real numbers v_i, b_i and $\mathbb{R}^p$ -vectors $\mathbf{w}_i$ such that, if we define*

$$\hat{f}(\boldsymbol{x}) := \sum_{i=1}^{N} v_i \phi(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)$$

then we have

$$\forall \mathbf{x} \in K_p, |\hat{f}(\mathbf{x}) - f(\mathbf{x})| \leq \epsilon.$$

From a theoretical perspective, this theorem is intriguing. Practically speaking, this is not very helpful due to the hidden layer's potential abundance of neurons. The depth in number of hidden layers of the networks is the strength of deep learning.

2.2.4 Estimation of the parameters

Following the selection of the network's architecture, a training data set must be used to estimate the parameters (the weights **W** and biases **b**). As usual, the estimation is performed by using a gradient descent technique to minimize a loss function. First, we must select the loss function.

Loss function for classification

The traditional method of parameter estimation is to maximize likelihood (or equivalently the logarithm of the likelihood). This corresponds to the minimization of the loss function which is the negative of the log likelihood. Denoting $\boldsymbol{\theta}$ the vector of parameters to estimate, we consider the expected loss function.

$$\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}) = -\mathbb{E}(\log(P(Y|\mathbf{x}, \boldsymbol{\theta})),$$

where $P(.|\mathbf{x}, \boldsymbol{\theta})$ is the probability mass function of Y given input $\mathbf{X} = \mathbf{x}$
For binary classification, with $Y \in \{0, 1\}$, maximizing the log likelihood corresponds to the minimization of the cross-entropy. Setting $p(\mathbf{x}, \boldsymbol{\theta})) = P(Y = 1|\mathbf{x}, \boldsymbol{\theta}) \in (0, 1)$,

$$\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}) = -\mathbb{E}[Y \log(p(\mathbf{x}, \boldsymbol{\theta})) + (1 - Y) \log(1 - p(\mathbf{x}, \boldsymbol{\theta}))].$$

The sigmoid activation function and this loss function work well together because the use of the logarithm prevents the gradient of $\mathcal{L}(\boldsymbol{\theta}, \mathbf{x})$ from having too small values.

Penalized empirical risk

For the estimation of the parameters $\boldsymbol{\theta}$, a training sample $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$ will be used. In particular, we mininize the empirical loss defined by

$$\hat{\mathcal{L}}_n(\boldsymbol{\theta}, \mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} (\ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})),$$

where $\ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta}))$ is the empirical loss for observation i when predicting $f(\mathbf{x}_i, \boldsymbol{\theta})$ while the actual answer is y_i in the training sample.

Backpropagation algorithm for the binary classification with the cross entropy

The binary classification case will be considered and explained how to compute the gradient of the empirical loss by the backpropagation algorithm. The output for the binary classification problem is

$$\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}) = \begin{pmatrix} P(Y = 0|\mathbf{x}, \boldsymbol{\theta}) \in (0,1) \\ P(Y = 1|\mathbf{x}, \boldsymbol{\theta}) \in (0,1) \end{pmatrix} = \begin{pmatrix} 1 - \phi_s(a^{L+1}) \\ \phi_s(a^{L+1}) \end{pmatrix}$$

We assume that the output activation function ψ is the sigmoid function ϕ_s, already defined in (4). Let's make a useful computation to compute the gradient of sigmoid function ϕ_s.

$$\frac{\partial \phi_s(x)}{\partial x} = \phi_s(x)(1 - \phi_s(x)).$$

We introduce the notation

$$(p(\mathbf{x}, \boldsymbol{\theta}))_y = \sum_{c=0}^{1} \mathbb{1}_{y=c}(p(\mathbf{x}, \boldsymbol{\theta}))_c,$$

where $(p(\mathbf{x}, \boldsymbol{\theta}))_1 = P(Y = 1|\mathbf{x}, \boldsymbol{\theta})$ and $(p(\mathbf{x}, \boldsymbol{\theta}))_0 = P(Y = 0|\mathbf{x}, \boldsymbol{\theta})$. Then we have

$$\ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = -\log(p(\mathbf{x}, \boldsymbol{\theta}))_y = -\sum_{c=0}^{1} \mathbb{1}_{y=c} \log(p(\mathbf{x}, \boldsymbol{\theta}))_c$$

for the loss function ℓ associated to the cross-entropy. Using the notations of Section 2.2.2, we want to compute the gradients

$$\text{Output weight} \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{W}^{(L+1)}} \qquad \text{Output biases} \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial b^{(L+1)}}$$

$$\text{Hidden weight} \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{W}^{(k)}} \qquad \text{Hidden biases} \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{b}^{(k)}}$$

for $1 \leq k \leq L$. We use the chain-rule: if $z(x) = \phi(a(x))$, then

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial a}\frac{\partial a}{\partial x}$$

Therefore

$$\frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial a^{(L+1)}} = \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial (p(\mathbf{x}, \boldsymbol{\theta}))_y} \frac{\partial (p(\mathbf{x}, \boldsymbol{\theta}))_y}{\partial a^{(L+1)}}$$

$$\text{we have, } \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial (p(\mathbf{x}, \boldsymbol{\theta}))_y} = \frac{-1}{(p(\mathbf{x}, \boldsymbol{\theta}))_y}$$

$$\frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial a^{(L+1)}} = \frac{-1}{(p(\mathbf{x}, \boldsymbol{\theta}))_1} \frac{\partial \phi_s(a^{(L+1)})}{\partial a^{(L+1)}} \mathbb{1}_{y=1}$$

$$
\begin{aligned}
&+ \frac{-1}{(p(\mathbf{x},\boldsymbol{\theta}))_0} \frac{\partial(1-\phi_s(a^{(L+1)}))}{\partial a^{(L+1)}} \mathbb{1}_{y=0} \\
&= \frac{-1}{(p(\mathbf{x},\boldsymbol{\theta}))_1} \phi_s(a^{(L+1)})(1-\phi_s(a^{(L+1)}))\mathbb{1}_{y=1} \\
&+ \frac{-1}{(p(\mathbf{x},\boldsymbol{\theta}))_0} \phi_s(a^{(L+1)})(-1+\phi_s(a^{(L+1)}))\mathbb{1}_{y=0} \\
&= (-1+(p(\mathbf{x},\boldsymbol{\theta}))_1)\mathbb{1}_{y=1} + (p(\mathbf{x},\boldsymbol{\theta}))_0\mathbb{1}_{y=0}
\end{aligned}
$$

Therefore we obtain

$$\nabla_{a^{(L+1)}} \ell(y, \mathbf{p}(\mathbf{x},\boldsymbol{\theta})) = \mathbf{p}(\mathbf{x},\boldsymbol{\theta}) - e(y).$$

where $y \in \{0,1\}$ and $e(y) = \mathbb{1}_{y=1}$. We now obtain easily the partial derivative of the loss function with respect to the output bias $b^{(L+1)}$.

$$\frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial b^{(L+1)}} = \frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial (p(\mathbf{x},\boldsymbol{\theta}))_y} \frac{\partial (p(\mathbf{x},\boldsymbol{\theta}))_y}{\partial a^{(L+1)}} \frac{\partial a^{(L+1)}}{\partial b^{(L+1)}}$$

Since

$$\frac{\partial a^{(L+1)}}{\partial b^{(L+1)}} = 1$$

Therefore

$$\nabla_{b^{(L+1)}} \ell(y, \mathbf{p}(\mathbf{x},\boldsymbol{\theta})) = \mathbf{p}(\mathbf{x},\boldsymbol{\theta}) - e(y).$$

Let us now compute the partial derivative of the loss function with respect to the output weights

$$\frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial \mathbf{W}^{(L+1)}} = \frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial a^{(L+1)}} \frac{\partial a^{(L+1)}}{\partial \mathbf{W}^{(L+1)}}$$

and

$$\frac{\partial a^{(L+1)}}{\partial \mathbf{W}^{(L+1)}} = \mathbf{h}^{(L)}$$

Therefore

$$\nabla_{\mathbf{W}^{(L+1)}} \ell(y, \mathbf{p}(\mathbf{x},\boldsymbol{\theta})) = (\mathbf{p}(\mathbf{x},\boldsymbol{\theta}) - e(y))\mathbf{h}^{(L)}.$$

Let us now compute the gradient of the loss function at hidden layers k. We use the chain rule

$$\frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial \mathbf{h}^{(k-1)}} = \frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial \mathbf{a}^{(k)}} \frac{\partial \mathbf{a}^{(k)}}{\partial \mathbf{h}^{(k-1)}}$$

We recall that

$$\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$$

Therefore

$$\frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial \mathbf{h}^{(k-1)}} = \frac{\partial \ell(y,\mathbf{p}(\mathbf{x},\boldsymbol{\theta}))}{\partial \mathbf{a}^{(k)}} \mathbf{W}^{(k)}$$

$$\nabla_{\mathbf{h}^{(k-1)}} \ell(y, \mathbf{p}(\mathbf{x},\boldsymbol{\theta})) = \mathbf{W}^{(k)} \ \nabla_{\mathbf{a}^{(k)}} \ \ell(y, \mathbf{p}(\mathbf{x},\boldsymbol{\theta})).$$

Recalling that $\mathbf{h}^{(k)} = \phi(\mathbf{a}^{(k)})$,

$$\frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{a}^{(k)}} = \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{h}^{(k)}} \phi'(\mathbf{a}^{(k)})$$

Therefore,

$$\nabla_{\mathbf{a}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \nabla_{\mathbf{h}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))(\phi'(\mathbf{a}^{(k)})).$$

This leads to

$$\begin{aligned} \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{W}^{(k)}} &= \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{a}^{(k)}} \frac{\partial \mathbf{a}^{(k)}}{\partial \mathbf{W}^{(k)}} \\ &= \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{a}^{(k)}} \mathbf{h}^{(k-1)} \end{aligned}$$

Finally, the gradient of the loss function with respect to hidden weights is

$$\nabla_{\mathbf{W}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \nabla_{\mathbf{a}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) \mathbf{h}^{(k-1)}.$$

The last step is to compute the gradient with respect to the hidden biases. We simply have

$$\frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{b}^{(k)}} = \frac{\partial \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \mathbf{a}^{(k)}}$$

and

$$\nabla_{\mathbf{b}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \nabla_{\mathbf{a}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})).$$

We can now summarize the backpropagation algorithm.

- **Forward pass**: we fix the value of the current parameters $\boldsymbol{\theta}^{(r)} = (\mathbf{W}^{(1,r)}, \mathbf{b}^{(1,r)}, ..., \mathbf{W}^{(L+1,r)}, b^{(L+1,r)})$, and we compute the predicted values $f(X_i, \boldsymbol{\theta}^{(r)})$ and all the intermediate values $(\mathbf{a}^{(k,r)}, \mathbf{h}^{(k,r)}) = \phi(\mathbf{a}^{(k,r)}))_{1 \le k \le L+1}$ that are stored. In the following, we suppress the dependence on r and use $\mathbf{W}^{(k)}$, $\mathbf{a}^{(k)}$ and $\mathbf{h}^{(k)}$ to ease notation
- **Backpropagation algorithm**:
  - Compute the output gradient $\nabla_{\mathbf{a}^{(L+1)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}) - e(y)$,
  - For $k = L + 1$ to 1
  - Compute the gradient at the hidden layer k

$$\nabla_{\mathbf{W}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \nabla_{\mathbf{a}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) \mathbf{h}^{(k-1)}$$

$$\nabla_{\mathbf{b}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \nabla_{\mathbf{a}^{(k)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})).$$

  - Compute the gradient at the previous layer

$$\nabla_{\mathbf{h}^{(k-1)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \mathbf{W}^{(k)} \ \nabla_{\mathbf{a}^{(k)}} \ \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta}))$$

and

$$\nabla_{\mathbf{a}^{(k-1)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) = \nabla_{\mathbf{h}^{(k-1)}} \ell(y, \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})) \phi'(\mathbf{a}^{(k-1)}).$$

Initialization

Before training, all input data needs to be normalized so that its range will be the same approximately. Then, we initialize all parameters. We can initialize 0 to all the bias values. However, there are some rules for the weight initialization. Firstly, the weights must not be initialized to 0 since the derivative at 0 of the tanh activation function $\phi_t(x)$ is 0, a saddle point. Secondly, the weights must not be initialized with the same numbers, otherwise, all the neurons of a hidden layer would have the same updating behavior. We generally initialize the weights randomly: the values of $\mathbf{W}^{(k)}$ are $i.i.d.$ uniform on $[-d, d]$ with possibly $d = \frac{\sqrt{6}}{N_l + N_{l-1}}$ where N_l is the number of the hidden layers in the neural network. Another option is to initialize the weights with a normal distribution $\mathcal{N}(0, 0.01)$ (see Glorot and Bengio [2010]).

Optimization algorithm

In order to minimize the criterion $\hat{\mathcal{L}}_n(\boldsymbol{\theta}, \mathrm{x})$, we use a **stochastic gradient descent (SGD)** algorithm. In particular, the **backpropagation algorithm**, introduced by Rumelhart et al. [1988], will be used to compute the gradient.
The stochastic gradient descent algorithm performs the following steps:

- Initialization of $\boldsymbol{\theta} = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, ..., \mathbf{W}^{(L+1)}, b^{(L+1)})^T$.
- For N iterations:
  - For each training observation $(\mathbf{x}_i, y_i)$,

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \frac{1}{m} \sum_{i \in B} [\nabla_{\boldsymbol{\theta}} \ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta}))],$$

where ϵ is **learning rate**, m is a **batch size** and $\nabla_{\boldsymbol{\theta}} \ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta}))$ is the gradient of $\ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta}))$ at $\boldsymbol{\theta}$. The value of ϵ must not be too small to prevent optimization from becoming stuck at a local minimum and causing the convergence to happen very slowly. However, the network will bounce around an optimal point if its value is too high, failing to stabilize and converge. A traditional solution to this problem is to change its value during the training of the model. It is advised to start with a high value of ϵ, such as 0.1, and to reduce its value during the successive iterations. However, there is no universal guideline for adjusting the learning rate, thus the engineer's experience in observing how the loss function changes over time will be more useful in determining how to proceed.

Note that, in the algorithm presented above, when updating a parameter, we do not compute the gradient for the loss function using the entire training data at each iteration. Instead,

a subset m, called a **batch**, of training examples are randomly selected from $\{1, ..., n\}$ without replacement to compute the gradients, and the average of the m corresponding gradients is used to update the parameters. This procedure is called **batch learning**. An iteration over all the training examples is called an **epoch**. In deep learning, the number of epochs is also included in the parameter sets. The total number of iterations equals the number of epochs times the sample size n divided by m, the size of a batch. For example, if the batch size m is 1/10 times the sample size n, an epoch corresponds to 10 batches. The predetermined number nb of epochs are used in the process iterations. Another stopping rule, called **early stopping** is also employed: it considers a validation sample and terminates learning whenever the loss function for this validation sample stops dropping. Batch learning is utilized for computational purposes since, as we have seen, the backpropagation algorithm requires the storage of all intermediate values computed during the forward step in order to compute the gradient during the backward pass, which is not possible for large data sets, such as those containing millions of images. This is made even more difficult by the fact that deep networks must calibrate millions of parameters. The batch size m is also a parameter to calibrate. Smaller batches typically have better generalization characteristics. **On-line gradient descent** refers to a specific scenario involving batches of size 1. The lengthy computing time of this process is a drawback.
Let us summarize the classical **SGD** algorithm.

ALGORITHM 1 **Stochastic Gradient Descent** algorithm

- *Fix the parameters ϵ: learning rate, m : batch size, nb : number of epochs.*
- *For l = 1 to nb epochs*
  - *Take a random batch of size m without replacement in the learning sample: $(\boldsymbol{x}_i, y_i)_{i \in B_l}$.*
  - *Compute the gradients with the backpropagation algorithm*

$$\tilde{\nabla}_{\boldsymbol{\theta}} = \frac{1}{m} \sum_{i \in B_l} \nabla_{\boldsymbol{\theta}} \ell(y_i, f(\boldsymbol{x}_i, \boldsymbol{\theta})).$$

  - *Update the parameters*

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta}^{old} - \epsilon \tilde{\nabla}_{\boldsymbol{\theta}}.$$

Since the choice of the learning rate is delicate and very influential on the convergence of the SGD algorithm, variations of the algorithm have been proposed. They are less sensitive to the learning rate. The principle is to add a correction when we update the gradient, called **momentum**. For more details, see Polyak [1964], Nesterov [1983] and Sutskever et al. [2013].

2.3 Convolutional neural networks

In general, multilayer perceptrons are not well suited for some specific data types, such as images. In order to deal with images, we typically convert them to numeric vectors, which frequently results in the loss of some of the spatial information, including shapes. The extraction of variables from features served as the foundation for the image learning in the past, however, this method requires extensive expertise in the image processing. The **convolutional neural networks (CNN)** introduced by LeCun [1998] have revolutionized the image processing, and eliminated the need for human feature extraction. The CNN acts directly on matrices, or even on tensors for images with three RGB color channels. The CNN is now commonly utilized for a variety of applications, including object and face recognition, image segmentation, and classification.

2.3.1 Layers in the CNN

A convolutional neural network is composed by several layers: **convolutional layers**, **pooling layers** and **fully connected layers**.

Convolution layer

For 2-dimensional signals such as images, we consider the 2D-convolution

$$Z(i,j) = (\mathbf{K}_{(conv)} * \mathbf{I})(i,j) = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \mathbf{I}(i+k_1-1, j+k_2-1)\mathbf{K}_{(conv)}(k_1,k_2),$$

where $\mathbf{K}_{(conv)} \in \mathbb{R}^{n_1 \times n_2}$ is a matrix convolution kernel applied to a 2D signal (or image) matrix $\mathbf{I} \in \mathbb{R}^{m_1 \times m_2}$. As shown in Figure 2.5, the principle of the 2D convolution is to drag a convolution kernel on the image. At each position, we get the convolution between the kernel and the part of the image currently treated. Then, the kernel moves by a number s of pixels, s is called the **stride**. When the stride is small, we get redundant information. Sometimes, we also add a **zero padding**, which is a margin of size p containing zero values around the image in order to control the size of the output.
Assume that we apply a kernel (also called filters) $\mathbf{K}_{(conv)}$ with the size of $n_1 \times n_2$ on an image with the size $m_1 \times m_2$, the size of the convolution output is $W_0 \times H_0$, where

$$W_0 = \frac{m_1 - n_1 + 2p}{s} + 1$$
$$H_0 = \frac{m_2 - n_2 + 2p}{s} + 1.$$

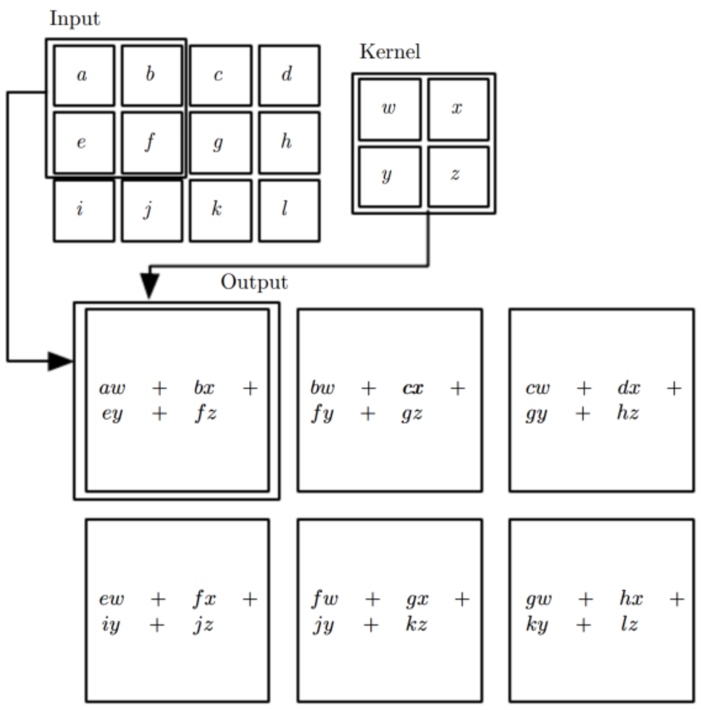


Figure 2.5: 2D convolution with the stride **s** = 1 and no padding. (Source: https://i2.wp.com/khshim.files.wordpress.com).

The convolution operations are combined with an activation function ϕ (generally the ReLu activation function ϕ_{ReLu}). The activation is obtained by sliding is:

$$A(i,j) = \phi([\sum_{k_1=1}^{n_1}\sum_{k_2=1}^{n_2} \mathbf{I}(i+k_1-1, j+k_2-1)\mathbf{K}_{(conv)}(k_1,k_2)] + \mathbf{b}_{(conv)}),$$

where $\mathbf{b}_{(conv)} \in \mathbb{R}^{n_1 \times 1}$ is a convolution bias.

It is in the convolution layer that we find the power of the CNN. Indeed, the CNN will learn the filters (or kernels) that are the most useful to the task that we have to do (such as classification). Another advantage is that several convolution layers can be considered: the output of a convolution becomes the input of the next one.

Pooling layer

Additionally, the CNN has pooling layers that enable dimension reduction, also known as subsampling, by taking the mean or the maximum on the selected picture patches (**mean-pooling** or **max-pooling**). Pooling layers act on small portions of the image similarly to convolutional layers, we also have a stride. If we consider 2 × 2 patches (pooling size = 2), over which we take the maximum value to define the output layer, and a stride $s = 2$, we divide by 2 the width

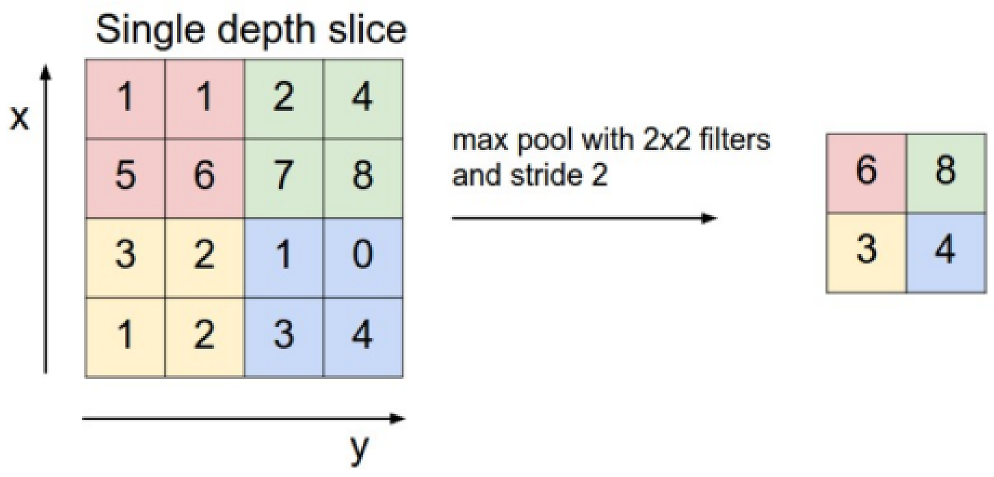


Figure 2.6: Maxpooling and effect on the dimension. Only the maximum for each color is maintained. (Source: http://www.wildml.com).

and height of the image. In general, the dimensions of output obtained after applying a pooling layer on an input with the size of $m_1 \times m_2$ is

$$W_0 = \frac{m_1 - R}{s} + 1$$
$$H_0 = \frac{m_2 - R}{s} + 1,$$

where s is the stride and R is the pooling size. Figure 2.6 represents an example of the max pooling.

Of course, it is also possible to reduce the dimension with the convolutional layer, by taking a stride larger than 1, and without zero padding, but another advantage of the pooling is that it makes the network less sensitive to small translations of the input images.

Fully connected layers for binary classification

The CNN generally finishes with one or more fully connected layers after numerous convolution and pooling layers. Additionally, fully connected layers vectorize and concatenate the output of last pooling layer as input (see Figure 2.7). Let **z** is the output after concatenation and vectorization,

$$\mathbf{z} = C(f_v(P_{out(i)})_{i=1,..,L_p}),$$

where P_{out} is the last pooling layer with size of $p_1 \times p_2$ for each output channel, L_p is the number of output channels in the last pooling layer, f_v is the function for vectorization and C is the function for concatenation. The above equation denotes the process of vectorizing the

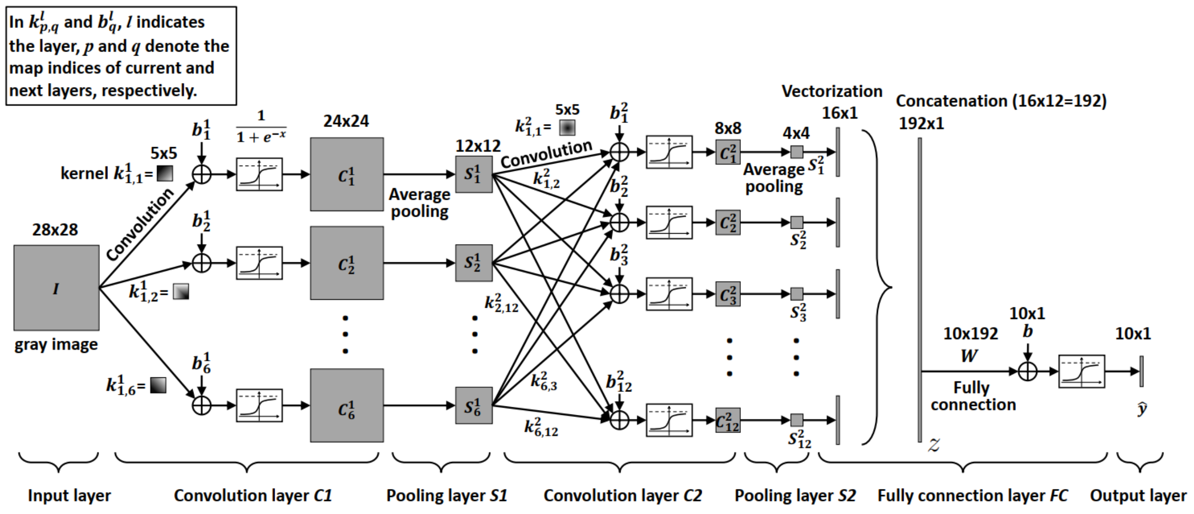


Figure 2.7: The CNN structure used in Zhifei [2016].

output of each channel by column scan and concatenating them to form a whole string. Then, the whole string is the input into the activation function for predicting the class labels.

$$\hat{\mathbf{p}} = \begin{pmatrix} \hat{p_0} \\ \hat{p_1} \end{pmatrix} = \phi_s(\mathbf{W}^{(L+1)}\mathbf{z} + b^{(L+1)}),$$

where $\hat{p_c} = \hat{P}(Y = c|\mathbf{x})$ for $c \in \{0,1\}$, $\mathbf{W}^{(L+1)} \in \mathbb{R}^{1\times(p_1 \times p_2 \times L)}$, $b^{(L+1)} \in \mathbb{R}$ and ϕ_s is softmax function.

Backpropagation in CNN

We choose cross-entropy as the loss function. The cross-entropy loss is specified as below:

$$\ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})) = -\sum_{c=0}^{1} \mathbb{1}_{y_i=c}\log(\hat{p_c})$$

Similar to a neural network, we still begin with performing forward propagation to generate $\hat{\mathbf{p}}$ with initial value of the weight and bias in convolution layer and fully connection layer. Then we resort to the gradient descent to update the weights and bias from fully connection layer to convolution layer.

For fully connection layer, we perform similar steps as in the backpropagation for the neutral network in Section 2.2.4.

$$\begin{aligned}\frac{\partial \ell}{\partial \mathbf{W}^{(L+1)}} &= \frac{\partial \ell}{\partial \hat{\mathbf{p}}}\frac{\partial \hat{\mathbf{p}}}{\partial \mathbf{W}^{(L+1)}} \\ &= (\hat{\mathbf{p}} - e(c))\frac{\partial a^{(L+1)}}{\partial \mathbf{W}^{(L+1)}}\end{aligned}$$

$$\begin{aligned} &= (\hat{\mathbf{p}} - e(c))\mathbf{z}. \\ \frac{\partial \ell}{\partial b^{(L+1)}} &= \frac{\partial \ell}{\partial \hat{\mathbf{p}}} \frac{\partial \hat{\mathbf{p}}}{\partial b^{(L+1)}} \\ &= (\hat{\mathbf{p}} - e(c)). \end{aligned}$$

where $e(c) = \mathbb{1}_{c=1}$.
For convolution layer with $\mathbf{K}_{(conv)} \in \mathbb{R}^{n_1 \times n_2}$ and $\mathbf{I} \in \mathbb{R}^{m_1 \times m_2}$, we have

$$\begin{aligned} \frac{\partial \ell}{\partial \mathbf{K}^{(k)}_{(conv)}(k_1,k_2)} &= \frac{\partial \ell}{\partial A(i,j)} \frac{\partial A(i,j)}{\partial \mathbf{K}^{(k)}_{(conv)}(k_1,k_2)} \\ &= \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} \frac{\partial \ell}{\partial A(i,j)} \frac{\partial(\phi([\sum_{k_1=1}^{n_1}\sum_{k_2=1}^{n_2} \mathbf{K}^{(k)}_{(conv)}(k_1,k_2)\mathbf{I}(i+k_1-1, j+k_2-1)] + \mathbf{b}^{(k)}_{(conv)}))}{\partial \mathbf{K}^{(k)}_{(conv)}(k_1,k_2)} \\ &= \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} \frac{\partial \ell}{\partial A(i,j)} A(i,j)(1-A(i,j))X(i-k_1, j-k_2) \\ &= \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} \alpha(i,j)X(i-k_1,j-k_2) \\ &= \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} \alpha * X, \end{aligned}$$

where $\alpha = \frac{\partial \ell}{\partial A(i,j)} A(i,j)(1-A(i,j))$ and $\frac{\partial \ell}{\partial A(i,j)}$ can be obtained by taking the derivative of the vectorized matrix and reorganizing as a matrix $\frac{\partial \ell}{\mathbf{z}}$ by using a size $p_1 \times p_2$ of the last pooling layer P_{out},

$$\begin{aligned} \frac{\partial \ell}{\mathbf{z}} &= \frac{\partial \ell}{\partial \hat{\mathbf{p}}} \frac{\partial \hat{\mathbf{p}}}{\mathbf{z}} \\ &= -\frac{1}{\hat{\mathbf{p}}}\mathbf{W} \\ \frac{\partial \ell}{\partial P_{out}} &= C^{-1}(\frac{\partial \ell}{\partial \mathbf{z}}) \\ \frac{\partial \ell}{\partial A(i,j)} &= p_1 p_2 \frac{\partial \ell}{\partial P_{out}}([\frac{i}{p_1}, \frac{j}{p_2}]). \end{aligned}$$

$$\begin{aligned} \frac{\partial \ell}{\partial \mathbf{b}^{(k)}_{(conv)}} &= \frac{\partial \ell}{\partial A(i,j)} \frac{\partial A(i,j)}{\partial \mathbf{b}^{(k)}_{(conv)}} \\ &= \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} \frac{\partial \ell}{\partial A(i,j)} \times 1 \\ &= \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} p_1 p_2 \frac{\partial \ell}{\partial P_{out}}([\frac{i}{p_1}, \frac{j}{p_2}]). \end{aligned}$$

Then, we update these parameters as mentioned in Section 2.2.4.

$$\mathbf{W}^{(L+1)} = \mathbf{W}^{(L+1)} - \epsilon \frac{\partial \ell}{\partial \mathbf{W}^{(L+1)}}$$

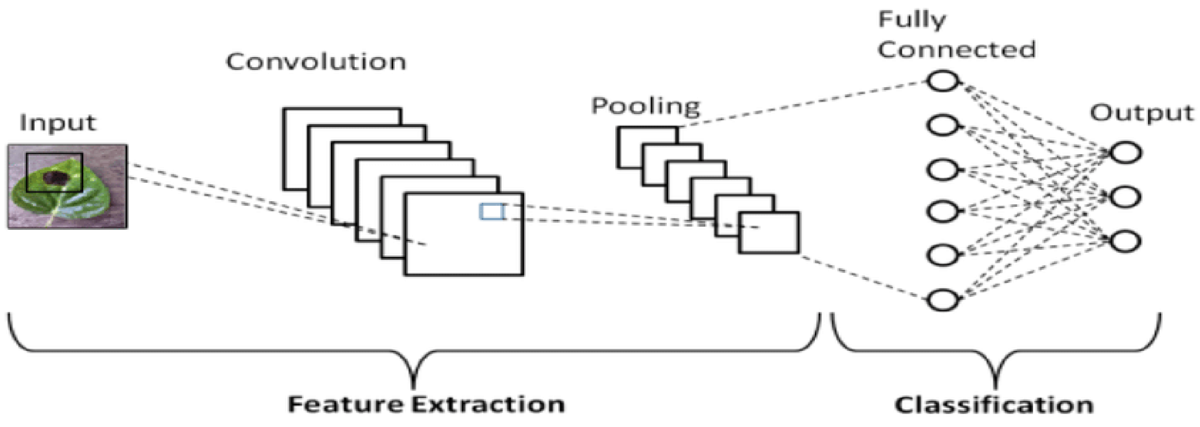


Figure 2.8: Architecture of a classical convolutional neural network (CNN).

$$
\begin{aligned}
b^{(L+1)} &= b^{(L+1)} - \epsilon \frac{\partial \ell}{\partial b^{(L+1)}} \\
\mathbf{K}^{(k)}_{(conv)}(k_1, k_2) &= \mathbf{K}^{(k)}_{(conv)}(k_1, k_2) - \epsilon \frac{\partial \ell}{\partial \mathbf{K}^{(k)}_{(conv)}(k_1, k_2)} \\
\mathbf{b}^{(k)}_{(conv)} &= \mathbf{b}^{(k)}_{(conv)} - \epsilon \frac{\partial \ell}{\partial \mathbf{b}^{(k)}_{(conv)}}
\end{aligned}
$$

where ϵ is the **learning rate**.

2.3.2 Architectures of the CNN

We have described the different types of layers included in the CNN. In order to design a network, we need to arrange these layers properly. However, designing an architecture is very complicated. Therefore, we only study the architectures which have proved to be effective and powerful. The classical CNN models usually start with an input layer, then some convolution layers are added to be followed by a pooling layer and end with a fully connected layer (see Figure 2.8).

In 2014, a network, **GoogLeNet** (Szegedy, Ioffe, Vanhouche and Alemi [2016]), was the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It is a new kind of the CNN, not only composed of the successive convolution and pooling layers, but also on the new modules, called **inception**, which are some kind of network in the network. An example is represented in Figure 2.9.

The most recent innovation is the **ResNet** network (Szegedy, Ioffe, Vanhouche and Alemi [2016]). The idea of the **ResNet** is to add a connection linking the input of a layer (or a set of layers) with its output. The **GoogleNet** and **ResNet** are much deeper than the previous CNN, but contain much fewer parameters. They are nevertheless more costly in memory than the classical

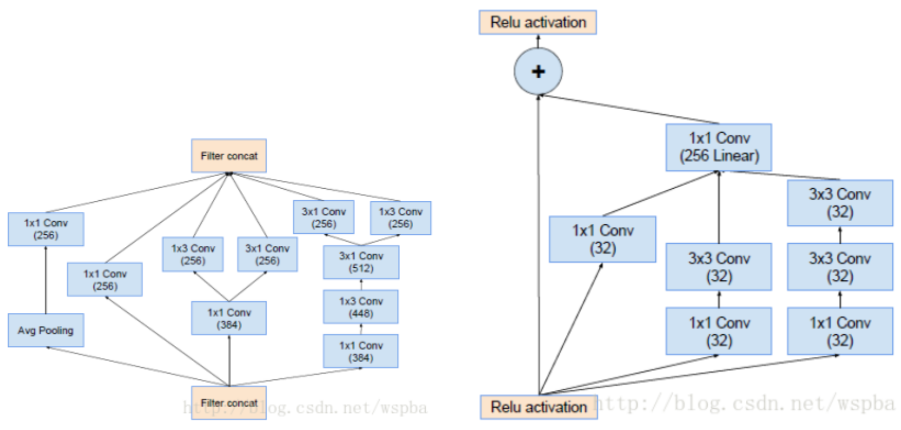


Figure 2.9: Inception modules in Szegedy, Ioffe, Vanhouche and Alemi [2016].

CNN. My thesis utilizes the **ResNet** network.

2.4 Residual network (ResNet)

The **ResNet** is a special and outstanding CNN network. Its model size is bigger than the **MobileNet** or **LeNet**. It was firstly launched in 2015 in Kaiming et al. [2015] and very soon to gain the first rank on the ILSVRC 2015.

The idea of the **ResNet** is that we construct a CNN architecture, then we add batch normalization and skip connection to each block. It allows us to tune the model's depth according to our requirement as flexible as possible. Some variations of the **ResNet** depth version are **ResNet18, ResNet34, ResNet50, ResNet101, ResNet152**. The main difference between these variations is the number of blocks and such blocks are stacked in side by side from the start to the end, which enables us to adjust the output shape being gradually smaller.

2.4.1 Batch normalization

The **ResNet** is the very first CNN architecture which applies batch normalization inside each block. Batch normalization helps to keep gradient descent algorithms stable and fastens the training process convergence to the optimal point.

Batch normalization is applied on each mini-batch by standard $\mathcal{N}(0,1)$ normalization. For example, given the input **x** over a mini-batch of size m, $B_i = \{x_1, x_2, ..., x_m\}$. All the input samples are re-scaled as follows:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}, \text{ where}$$

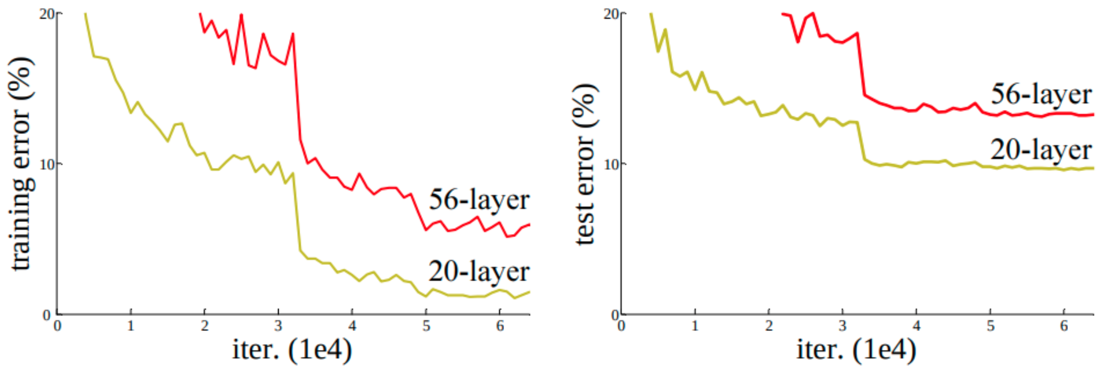


Figure 2.10: Training error (left) and test error (right) on the CIFAR-10 data set with 20-layer and 56-layer plain networks. The deeper network has higher training error, and thus test error. (Source: https://arxiv.org/pdf/1512.03385.pdf).

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$
$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

It is usually used for $m = 128$ or 256.

2.4.2 Skip connection

Szegedy, Ioffe, Vanhouche and Alemi [2016] thoroughly studied the efficiency of the changes in depth to the model accuracy. Actually, when the model depth increases, we meet accuracy saturation, further increasing in the depth may lead to degradation (Figure 2.10). From this evidence state, it follows that in order to improve the model accuracy, it is not simply make it deeper.

Therefore, Szegedy, Ioffe, Vanhouche and Alemi [2016] used a deep residual learning framework as a main solution. They intentionally allowed these layers to fit a residual mapping rather than hoping that each few stacked layer would directly fit a desired underlying mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(\mathbf{x})$, the authors let the stacked nonlinear layers fit another mapping of $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original mapping is recast as $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. According to their theory, optimizing the residual mapping is simpler than optimizing the original and unreferenced mapping. In the extreme, if an identity mapping was optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers. The formulation of $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ can be realized by a feedforward neural network with the **shortcut connections** (Figure 2.11). The shortcut connections are those skipping one or more layers. In

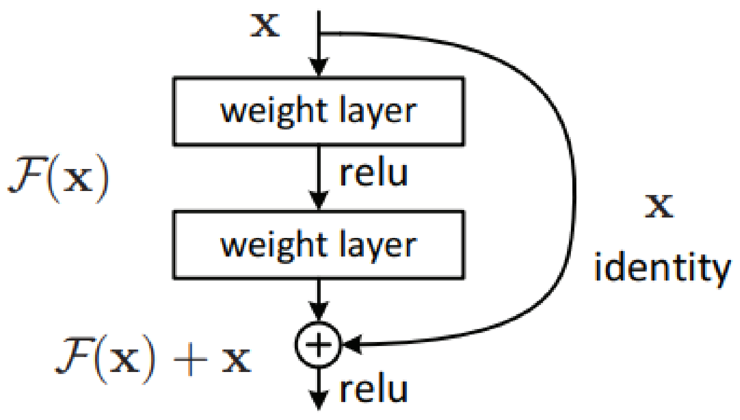

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity

Figure 2.11: Residual learning: a building block.

our case, the shortcut connections simply perform the identity mapping, and their outputs are added to the outputs of the stacked layers. More generally,

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{\mathbf{W}^{(k)}\}) + \mathbf{x},$$

Here, $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{\mathbf{W}^{(k)}\})$ represents the residual mapping to be learned. The identity shortcut connections add neither extra parameter nor computational complexity. The entire network can still be trained end-to-end by the SGD with backpropagation.

Another step they applied is a convolutional mapping before the skip connection from input layers to output layers in order to allow feature learning,

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{W}^{(k)}) + Conv(\mathbf{x}).$$

To keep the shape of output unchanged and reduce the total parameters, convolution mapping normally has the kernel size of 1×1 (Figure 2.12).

2.4.3 Network architecture

In this thesis, we will choose the ResNet50 architecture as a main method to train our image data set. This architecture is represented in Figure 2.13. In particular, we assume

- Zero-padding pads of the input with size of (3,3)
- Stage 1
    - The 2D convolution has 64 filters of shape (7,7) and uses a stride = 2.

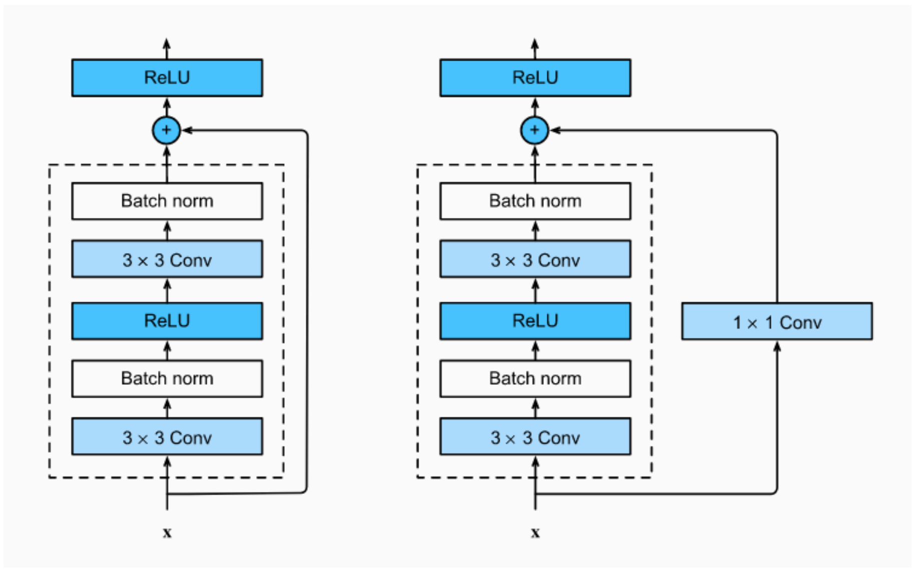


Figure 2.12: A regular block (left) and a residual block (right).

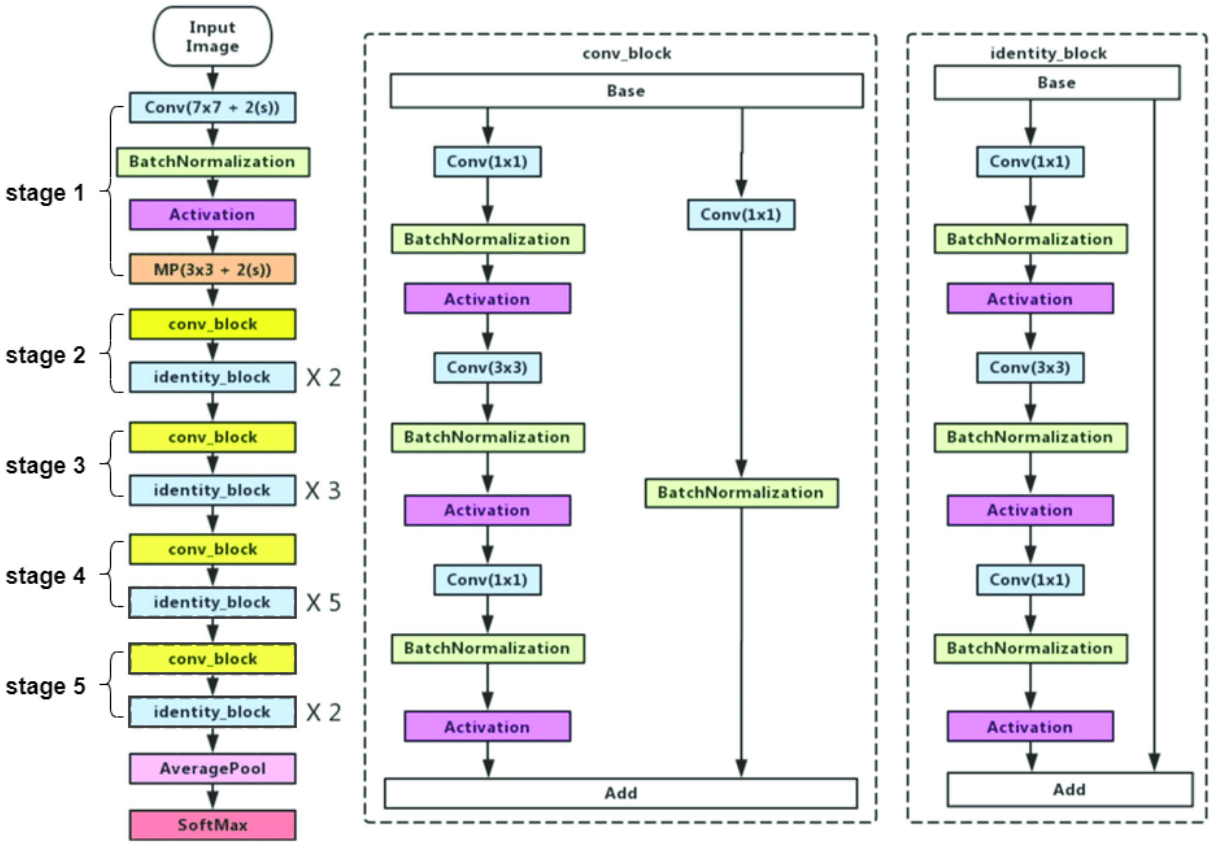


Figure 2.13: The ResNet50 architecture. “X 3” means we stack 3 blocks together.

  - The batchnorm is applied to the channels axis of the input.
  - The maxpooling uses a window with size of (3,3) and a stride with size of 2.

- Stage 2
  - The convolutional block uses three set of filters of size [64,64,256] with the shape (3,3) and a stride with size of 1 for each filter.
  - The two identity blocks use three set of filters of size [64,64,256] with the shape (3,3) for each filter.

- Stage 3
  - The convolutional block uses three set of filters of size [128,128,512] with the shape (3,3) and a stride with size of 2 for each filter.
  - The three identity blocks use three set of filters of size [128,128,512] with the shape (3,3) for each filter.

- Stage 4
  - The convolutional block uses three set of filters of size [256, 256, 1024] with the shape (3,3) and a stride with size of 2 for each filter.
  - The five identity blocks use three set of filters of size [256, 256, 1024] with the shape (3,3) for each filter.

- Stage 5
  - The convolutional block uses three set of filters of size [512, 512, 2048] with the shape (3,3) and a stride with size of 2 for each filter.
  - The two identity blocks use three set of filters of size [512, 512, 2048] with the shape (3,3) for each filter.

- The 2D average pooling uses a window of shape (2,2).

- The fully connected layer reduces its input to the number of classes using a softmax activation ϕ_s.

2.5 SHAP

The SHAP (SHapley Additive exPlanations) was developed by (Scott and Lee [2017]), based on the game theoretically optimal Shapley values (Shapley [1953]). The SHAP is a game theoretic approach to explain the output of machine learning models by computing the Shapley values for each feature. The SHAP applies the main idea of coalition game theory (Roger [1991]), where we have players, a game and a payout. The purpose of game theory is to distribute the payout between the players based on their contributions.

2.5.1 Shapley value and cooperative game theory

Consider a cooperative game with M players aiming at maximizing a payoff, and let $\mathcal{S} \subseteq \mathcal{M} = \{1, 2, ..., M\}$ be a subset of the features in $\mathcal{M}$. Assume that we have a function $v(\mathcal{S})$ that maps subsets of players to the real numbers, called the **contribution** of the coalition $\mathcal{S}$. It explains the total expected sum of payoffs that the members of $\mathcal{S}$ can obtain through collaboration. The **Shapley value** (Shapley [1953]) is one method to distribute the total payout to the players, given that everyone cooperates. According to the Shapley value, the amount that player j gets (or the **contribution of player** j) is

$$\phi_j(v) = \phi_j = \sum_{\mathcal{S} \subseteq \mathcal{M} \setminus \{j\}} \frac{|\mathcal{S}|!(M - |\mathcal{S}| - 1)!}{M!} (v(\mathcal{S} \cup \{j\}) - v(\mathcal{S})), j = 1, ..., M, \tag{5}$$

Keep in mind that this sum also includes the empty set $\mathcal{S} = \varnothing$. The following is how the formula can be understood: imagine the coalition being established for one player at a time, with each player demanding their contribution $v(\mathcal{S} \cup \{j\}) - v(\mathcal{S})$ as a fair compensation. Then, for each player, compute the average of this contribution over all possible combinations in which the coalition can be formed, yielding the weighted mean given in (5).

To illustrate the application of (5), let us consider a game with three players such that $M = \{2, 3, 4\}$. Then, there are 8 possible subsets; $\varnothing, \{2\}, \{3\}, \{4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}$, and $\{2, 3, 4\}$. Using (5), the Shapley values for the three players are given by

$$\phi_2 = \frac{1}{3}(v(\{2,3,4\}) - v(\{3,4\}) + \frac{1}{6}(v(\{2,3\}) - v(\{3\}) + \frac{1}{6}(v(\{2,4\}) - v(\{4\}) + \frac{1}{3}(v(\{2\}) - v(\{\varnothing\})$$
$$\phi_3 = \frac{1}{3}(v(\{2,3,4\}) - v(\{2,4\}) + \frac{1}{6}(v(\{2,3\}) - v(\{2\}) + \frac{1}{6}(v(\{3,4\}) - v(\{4\}) + \frac{1}{3}(v(\{3\}) - v(\{\varnothing\})$$
$$\phi_4 = \frac{1}{3}(v(\{2,3,4\}) - v(\{2,3\}) + \frac{1}{6}(v(\{2,4\}) - v(\{2\}) + \frac{1}{6}(v(\{3,4\}) - v(\{3\}) + \frac{1}{3}(v(\{4\}) - v(\{\varnothing\}).$$

Let's define the non-distributed gain $\phi_0 = v(\varnothing)$, that is, the fixed payoff which is not associated with any activities of the players, although this is often zero for coalition games. By using the right hand side above, it is clear to us that they add up to the total worth of the game: $\phi_0 + \phi_2 + \phi_3 + \phi_4 = v(\{2, 3, 4\})$

2.5.2 Shapley value for prediction explanation

Consider a classical machine learning model where our training data set has n observations with p features

$$(y_i, \mathbf{x}_i^T)^T \coloneqq (y_i, x_{i1}, x_{i2}, ..., x_{ip})^T, i = 1, ..., n,$$

We use this data set to train a predictive model denoted by $\hat{f}(\mathbf{x})$ attempting to resemble a response value y as closely as possible. Assume now that we want to interpret the prediction for an observation i from the model. Strumbel and Kononenko [2010] suggest doing this using the Shapley value. By moving from game theory to decomposing an individual prediction into feature contributions, the single prediction corresponds to the payout, and the features take the place of the players $(M = p)$. We have that the prediction $\hat{f}(\mathbf{x}_i)$ at $\mathbf{x}_i$ is decomposed as follows:

$$\hat{f}(\mathbf{x}_i) = \phi_0 + \sum_{j=1}^{M} \phi_{ij},$$

where $\phi_0 = \beta_0 + \sum_{j=1}^{p} \beta_j E(X_j) = E_{\mathbf{X}}(f(\mathbf{X}))$ if $\hat{f}$ is linear in $\mathbf{X}$, ϕ_{ij} is the contribution of feature j for the prediction at $\mathbf{x}_i$ and M is the number of players (features). That is, the Shapley value explains the difference between the prediction $\hat{f}(\mathbf{x}_i)$ and the global average prediction. A model of this form is an additive feature attribution method.

Suppose we have a linear model and we want to explain the prediction for an observation $\mathbf{x}_i$ with p predictors. The prediction for $\mathbf{x}_i$ is

$$\hat{f}(\mathbf{x}_i) = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + ... + \hat{\beta}_p x_{ip}.$$

By considering p predictors as M players and the prediction is the payout. Let ϕ_{ij} be a contribution of predictor j on $\hat{f}(\mathbf{x}_i)$. According to Kjersti, Martin and Anders [2019], the Shapley value takes the simple form:

$$\hat{\phi}_0 = \hat{\beta}_0 + \sum_{j=1}^{M} \hat{\beta}_j E(X_j), \text{ and } \hat{\phi}_{ij} = \hat{\beta}_j (x_{ij} - E(X_j)), j = 1, ..., M \tag{6}$$

where $E(X_j)$ is the expected value for feature j. If all feature contributions are summed up, the result is

$$\begin{aligned}\sum_{j=1}^{M} \phi_{ij} &= \sum_{j=1}^{M} (\beta_j x_{ij} - E(\beta_j X_j)) \\ &= \left(\beta_0 + \sum_{j=1}^{M} \beta_j x_{ij}\right) - \left(\beta_0 + \sum_{j=1}^{M} E(\beta_j X_j)\right) \\ &= \hat{f}(\mathbf{x}_i) - \sum_{j=1}^{M} E(\beta_j X_j) - \beta_0\end{aligned}$$

$$=\hat{f}(\mathbf{x}_i) - E_{\mathbf{X}}(\hat{f}(\mathbf{X})).$$

This is the predicted value for the data point $\mathbf{x}_i$ minus the expected predicted value. Feature contributions can be negative. According to Kjersti, Martin and Anders [2019], no explicit formula like (6) exists for the general case of dependent features with non-linear models. Since the number of possible coalitions exponentially grows as more features are added, using (5) directly to determine the exact solution is exceedingly challenging. As a result, based on (5), Strumbel and Kononenko [2014] suggested an approximate approach to calculate the Shapley value.

$$\hat{\phi}_{ij} = \frac{1}{M'} \sum_{m'=1}^{M'} (\hat{f}(\mathbf{x}_{i,+j}^{m'}) - \hat{f}(\mathbf{x}_{i,-j}^{m'})), \tag{7}$$

where M' is chosen to be $\ll |\mathcal{S}|$ with $\mathcal{S} \subseteq \{1,2,...,M\} \setminus \{j\}$, $\hat{f}(\mathbf{x}_{i,+j}^{m'})$ is the prediction for record $\mathbf{x}_i$, but with a random number of feature values replaced by some feature values from a random data vector $\mathbf{z}$ in the original data set with the exception of the respective value for feature j. The prediction $\hat{f}(\mathbf{x}_{i,-j}^{m'})$ is identical to $\hat{f}(\mathbf{x}_{i,+j}^{m'})$ but the respective value of feature j is taken from a feature value of z. The steps of the approximate Shapley value estimation are represented as below.

Approximate Shapley value estimation for single feature value:

- Input: Number of iterations M', instance of interested observation $\mathbf{x}_i$, feature index j, original data matrix $\mathbf{X}$, and machine learning model $\hat{f}$.
- Output: Shapley value for the value of the j-th feature.
- For all $m' = 1, ..., M'$.
  - Draw random instance $\mathbf{z}$ from the data matrix $\mathbf{X}$.
  - Instance of interested observation $\mathbf{x}_i$: $\mathbf{x}_i = (x_{i1}, ..., x_{ij}, ..., x_{ip})$.
  - Instance $\mathbf{z}_i$: $\mathbf{z}_i = (z_{i1}, ..., z_{ij}, ..., z_{ip})$.
  - Construct two new instances.
    * $\mathbf{x}_{i,+j}^{m'} = (x_{i1}, ..., x_{i,j-1}, x_{ij}, z_{i,j+1}, ..., z_{ip})$.
    * $\mathbf{x}_{i,-j}^{m'} = (x_{i1}, ..., x_{i,j-1}, z_{i,j}, z_{i,j+1}, ..., z_{ip})$.
  - Compute marginal contribution $\hat{\phi}_{ij}^{m'} = \hat{f}(\mathbf{x}_{i,+j}^{m'}) - \hat{f}(\mathbf{x}_{i,-j}^{m'})$.
- Estimate the Shapley value as the average $\hat{\phi}_{ij}(\mathbf{x}_i) = \frac{1}{M'} \sum_{m=1}^{M'} \hat{\phi}_{ij}^{m'}$.

For images, by considering a player to be a pixel and prediction of the response is the payout. We can apply the above process to estimate the contribution of each pixel to the prediction of the response. In this way, we can identify which pixel has a positive/negative contribution to the prediction of the response depending on the sign of the Shapley value of the pixel.

Chapter 3

Residual network based credit scoring for two data sets

In this thesis, we used the two data sets, which are the **German Credit Data** (Hans [1995]) and the **Home Credit Data** (Kaggle [2018]), for our experiment for training the ResNet50 network for credit scoring. Besides, we used the logistic regression as a baseline model to compare the performance of our ResNet50 network.

For both methods, we performed a cross-validation with 5 folds and evaluated their performances by four metrics: AUC, Accuracy, Recall and Precision.

We implemented the network training in Python using the **keras** library and implemented the WOE transformation in R using the **scorecard** library. We trained our network on a computer with the CPU Intel Core i5-7300 HQ, RAM 8GB. On this CPU, our training process took 8 minutes for one epoch for the **German Credit Data** and 1 hour for one epoch for the **Home Credit Data**. With the 5 folds of the cross-validation, we used from 40 to 45 epoches for **German Credit Data** and from 25 to 30 epoches for **Home Credit Data**.

3.1 Evaluation metrics

Suppose we have a binary classification model and its prediction for client i is $\hat{p}_i = P(Y_i = 1|\mathbf{X}_i = \mathbf{x}_i)$ for a threshold c. Y_i is classified as 1 (positive) $\Leftrightarrow \hat{p}_i \geq c$.
Then, for a given threshold c, we have

$$TP^{(c)} = \text{number of true positives using threshold } c$$
$$FP^{(c)} = \text{number of false positives using threshold } c$$
$$TN^{(c)} = \text{number of true negatives using threshold } c$$

$FN^{(c)}$ = number of false negatives using threshold c

$P^{(c)}$ = number of positives in ground truth using threshold c

$N^{(c)}$ = number of negatives in ground truth using threshold c

True Positive Rate (TPR$^{(c)}$) = $\frac{TP^{(c)}}{TP^{(c)} + FN^{(c)}}$

False Positive Rate (FPR$^{(c)}$) = $\frac{FP^{(c)}}{FP^{(c)} + TN^{(c)}}$

Accuracy$^{(c)}$ = $\frac{TP^{(c)} + TN^{(c)}}{P^{(c)} + N^{(c)}}$

Recall$^{(c)}$ = $\frac{TP^{(c)}}{TP^{(c)} + FN^{(c)}}$

Precision$^{(c)}$ = $\frac{TP^{(c)}}{TP^{(c)} + FP^{(c)}}$

Accuracy is simply a ratio of correctly predicted observations to the total number of observations. It measures how correct our model predicts in general. **Precision** is the ratio of correctly predicted positive observations to the total predicted positive observations. It measures how correct, or precise, our model's positive predictions are. **Recall** is the ratio of correctly predicted positive observations to all actual positive observations. It measures how many of the actual positive instances we are able to be correctly predicted.

The ROC curve plots $TPR^{(c)}$ versus $FPR^{(c)}$ at different classification thresholds c. **AUC** measures the entire two-dimensional area underneath the entire ROC curve, and it is the measure of the ability of a classifier to distinguish between classes across all possible classification thresholds.

All these metrics are commonly used to evaluate binary classification models. The higher the value of these metrics, the better the model is.

3.2 Data

For two data sets, we applied some rules for data processing. Firstly, we only used variables with the missing rate $< 10\%$. Secondly, we only used predictive variables by selecting ones with the IV > 2%. Since for the **Home Credit Data**, the default rate is about 8%, we decided to increase this rate to 30% by decreasing the number of non-default observations. Table 3.1 gives a short overview to both data sets before and after applying the mentioned rules.

Table 3.1: Overview of the **German Credit Data** and the **Home Credit Data** before and after data preparation steps.

	German Credit (before)	**German Credit** (after)
sample size	1000	1000
non-defaults	700	700
defaults	300	300
default rate	30%	30%
variables	20	10
categorical	13	8
continuous	7	2
	Home Credit (before)	**Home Credit** (after)
sample size	307511	82750
non-defaults	282686	57925
defaults	24825	24825
default rate	8%	30%
variables	122	18
categorical	16	5
continuous	106	13

3.2.1 German Credit Data

This public data set is available on (https://archive.ics.uci.edu/ml/datasets/). It classifies 1000 applicants described by a set of attributes/variables as goods (non-default) or bads (default) and the percentage of bad clients is 30%. The 10 attributes for our training data are defined as:

- Attribute 1: `chk_acct` - status of existing checking account (qualitative)
  - A11 : ... ≤ 0 DM
  - A12 : 0 ≤ ... ≤ 200 DM
  - A13 : ... ≥ 200 DM / salary assignments for at least 1 year
  - A14 : no checking account
- Attribute 2: `age` - age in years (numerical)
- Attribute 3: `duration` - duration in month (numerical)
- Attribute 4: `credit_hist` - credit history (qualitative)
  - A30 : no credits taken/all credits paid back duly
  - A31 : all credits at this bank paid back duly
  - A32 : existing credits paid back duly till now
  - A33 : delay in paying off in the past
  - A34 : critical account/other credits existing (not at this bank)
- Attribute 5: `purpose` - purpose (qualitative)
  - A40 : car (new)
  - A41 : car (used)
  - A42 : furniture/equipment
  - A43 : radio/television
  - A44 : domestic appliances
  - A45 : repairs
  - A46 : education
  - A47 : vacation
  - A48 : retraining
  - A49 : business

  - A410 : others

- Attribute 6: `saving_acct` - savings account/bonds (qualitative)
  - A61 : ...< 100 DM
  - A62 : 100 ≤ ... < 500 DM
  - A63 : 500 ≤ ... < 1000 DM
  - A64 : ... ≥ 1000 DM
  - A65 : unknown/no savings account
- Attribute 7: `present_emp` - present employment since (qualitative)
  - A71 : unemployed
  - A72 : ... < 1 year
  - A73 : 1 ≤ ... < 4 years
  - A74 : 4 ≤ ... < 7 years
  - A75 : ... ≥ 7 years
- Attribute 8: `property` - property (qualitative)
  - A121 : real estate
  - A122 : if not A121 : building society savings agreement/life insurance
  - A123 : if not A121/A122 : car or other, not in attribute 6
  - A124 : unknown / no property
- Attribute 9: `other_install` - other installment plans (qualitative)
  - A141 : bank
  - A142 : stores
  - A143 : none
- Attribute 10: `housing` - housing (qualitative)
  - A151 : rent
  - A152 : own
  - A153 : for free.

3.2.2 Home Credit Data

This public dataset is available on the Kaggle (https://www.kaggle.com/competitions/home-credit-default-risk/overview). This data set classifies the repayment abilities (default and non-default) of thousands of clients. The 18 attributes for our training data are defined as:

- Attribute 1: `AMT_ANNUITY` - loan annuity (numerical)
- Attribute 2: `DAYS_BIRTH` - client's age in days at the time of application (numerical)
- Attribute 3: `DAYS_REGISTRATION` - how many days before the application did client change his registration (numerical)
- Attribute 4: `DAYS_ID_PUBLISH` - how many days before the application did client change the identity document with which he applied for the loan (numerical)
- Attribute 5: `DAYS_EMPLOYED` - how many days before the application the person started current employment (numerical)
- Attribute 6: `AMT_CREDIT` - credit amount of the loan (numerical)
- Attribute 7: `DAYS_LAST_PHONE_CHANGE` - how many days before application did client change phone (numerical)
- Attribute 8: `AMT_GOODS_PRICE` - for consumer loans, it is the price of the goods for which the loan is given (numerical)
- Attribute 9: `EXT_SOURCE_2` - normalized score from external data source (numerical)
- Attribute 10: `OCCUPATION_TYPE` - what kind of occupation does the client have (Cleaning staff, Sales staff,etc.) (qualitative)
- Attribute 11: `REGION_POPULATION_RELATIVE` - normalized population of region where client lives (higher number means the client lives in more populated region) (numerical)
- Attribute 12: `NAME_INCOME_TYPE` - client's income type (working, maternity leave,etc.) (qualitative)
- Attribute 13: `REGION_RATING_CLIENT_W_CITY` - our rating of the region where client lives with taking city into account (1,2,3) (qualitative)
- Attribute 14: `NAME_EDUCATION_TYPE` - level of highest education the client achieved (qualitative)

- Attribute 15: `REGION_RATING_CLIENT` - our rating of the region where client lives (1,2,3) (qualitative)
- Attribute 16: `AMT_INCOME_TOTAL` - income of the client (numerical)
- Attribute 17: `CODE_GENDER` - gender of the client (qualitative)
- Attribute 18: `REG_CITY_NOT_WORK_CITY` - flag if client's permanent address does not match work address (1 = different, 0 = same, at city level) (qualitative).

3.3 WOE calculation and transformation to the image format

3.3.1 WOE calculation

We applied the method presented in Section 2.1 to calculate the WOE for all variables in the data set. As an example, we are considering the `duration` variable in the training data in the first fold of cross-validation performed on the **German Credit Data**. Table 3.2 illustrates the results of the WOE calculation for the `duration` variable.

Table 3.2: Result of the WOE calculation for the `duration` variable (IV = 0.302) of the **German Credit Data**. The goods are non-defaults and the bads are defaults.

`duration` interval	Count	Count Prob	Good clients	Good Prob	Bad clients	Bad Prob	WOE	Bin.IV
[-Inf,8)	73	0.091	66	0.118	7	0.029	-1.403	0.125
[8,16)	271	0.339	206	0.368	65	0.270	-0.309	0.030
[16,32)	314	0.392	217	0.388	97	0.404	0.040	0.001
[32, Inf)	142	0.178	71	0.126	71	0.297	0.857	0.147
Total	800		560		240			0.302

The continuous `duration` variable is bucketed into 4 bins: [-Inf,8), [8,16), [16,32) and [32, Inf). Each bin contains at least 5% of data and there is no bin with 0 count of either bads (defaults) or goods (non-defaults). For each bin,

- We estimate the probability of good and bad clients falling into a bin by dividing the number of good/bad clients respectively of each bin by the total of number good/bad clients in the data. For example, the probability of good and bad clients of the `duration` variable in the bin $[\text{-Inf}, 8)$ are estimated as follows:

$$\hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)} | Y = 1) = \frac{7}{240} = 0.029$$
$$\hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)} | Y = 0) = \frac{66}{560} = 0.118.$$

- We estimate the probability of clients falling into a bin (probability of count) by dividing the number of clients of a bin by the total of number of clients in the data. For example, the probability of count of the `duration` variable in the bin $[\text{-Inf}, 8)$ are estimated as follows:

$$\hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)}) = \frac{73}{800} = 0.091.$$

- We take the log of the probability of bad over the probability of good of each bin to obtain the value of the WOE for the corresponding bin. For example, the estimated WOE of the `duration` variable for the bin [-Inf,8) is estimated as:

$$\widehat{WOE}_{[\text{-Inf},8)} = \log \frac{\hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)} | Y = 1)}{\hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)} | Y = 0)} = \log(\frac{0.029}{0.118}) = -1.403.$$

- We take the difference between the probability of bad and the probability of good then multiply this difference by the WOE of the corresponding bin to get the IV value. For example, the IV of the `duration` variable for the bin [-Inf,8) is estimated as:

$$\begin{aligned}\widehat{IV}_{\texttt{duration}_{[\text{-Inf},8)}} &= (\hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)} | Y = 1) - \hat{P}(X_{\texttt{duration}} \in B_{[\text{-Inf},8)} | Y = 0)) \times \widehat{WOE}_{[\text{-Inf},8)} \\ &= (0.029 - 0.118) \times -1.403 = 0.125.\end{aligned}$$

The IV of the `duration` variable is the sum of the IV of all bins and it equals 0.302.

Figure 3.1 represents the distribution of the `duration` variable and the WOE trend after being transformed by the WOE technique. Each column in this figure represents the probability of count of each bin and the blue line is the trend of WOE across all bins. We can clearly see that the WOE is showing an increasing trend. It means that the longer the duration is, the higher probability of a bad credit is.

We applied the WOE technique to transform all continuous and categorical variables into discrete categories. The list below is the details of each variable after the WOE transformation in the first fold of the cross-validation performed on **German Credit Data** and the fifth fold of the cross-validation performed on **Home Credit Data** respectively. Note that the list could change when we apply the WOE technique to different training data in other folds of cross-validation.

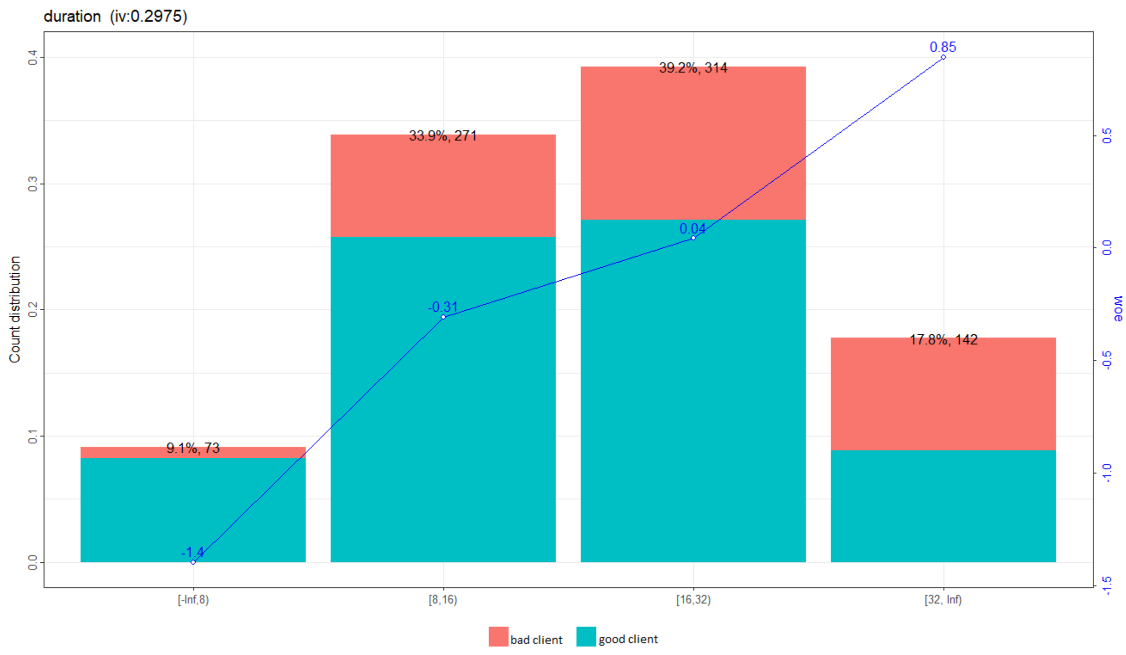


Figure 3.1: Distribution of `duration` variable after being transformed by the WOE technique.

German Credit Data

- Attribute 1: `chk_acct`
  1. No account (A14)
  2. ...< 200 DM (A12,A11)
  3. ...≥ 200 DM (A14)
- Attribute 2: `age`
  1. [-Inf,26)
  2. [26,35)
  3. [35,37)
  4. [37,53)
  5. [53,Inf)
- Attribute 3: `duration`
  1. [-Inf,8)
  2. [8,16)
  3. [16,32)

4. [32, Inf)

- Attribute 4: `credit_hist`
  1. critical account/other credits existing (A34)
  2. existing credits paid back duly till now (A32)
  3. delay in paying off in the past (A33)
  4. no credits taken/all credits paid back duly (A31,A30)
- Attribute 5: `saving_acct`
  1. ...≥ 500 DM (A64,A63,A65)
  2. 100 ≤...< 500 DM (A62)
  3. ...< 100 DM (A61)
- Attribute 6: `purpose`
  1. car(used) and retraining (A41,A48)
  2. radio/television and repairs (A43,A45)
  3. furniture/equipment (A42)
  4. business and domestic appliances (A49,A44)
  5. others, car(new) and education (A410,A40,A46)
- Attribute 7: `property`
  1. real estate (A121)
  2. building society savings agreement/life insurance (A122)
  3. car or other (A123)
  4. unknown / no property (A124)
- Attribute 8: `present_emp`
  1. ...≥ 7 years (A75)
  2. 4 ≤ ... < 7 years (A74)
  3. 1 ≤ ... < 4 years (A73)
  4. unemployed and ...< 1 (A72, A71)
- Attribute 9: `housing`

1. own (A152)
2. rent (A151)
3. for free (A153)

- Attribute 10: `other_install`

  1. none (A143)
  2. bank (A141)
  3. stores (A142).

Home Credit Data

- Attribute 1: `AMT_ANNUITY`

  1. missing
  2. [-Inf,16000)
  3. [16000,36000)
  4. [36000,44000)
  5. [44000, Inf)

- Attribute 2: `DAYS_BIRTH`

  1. [-Inf,-20000)
  2. [-20000,-17000)
  3. [-17000,-13500)
  4. [-13500, Inf)

- Attribute 3: `DAYS_REGISTRATION`

  1. [-Inf,-9000)
  2. [-9000,-7000)
  3. [-7000,-1000)
  4. [-1000, Inf)

- Attribute 4: `DAYS_ID_PUBLISH`

  1. [-Inf,-4200)
  2. [-4200,-3200)

3. [-3200,-1900)
4. [-1900, Inf)

- Attribute 5: `DAYS_EMPLOYED`

1. [-Inf,-4400)
2. [-4400,-3000)
3. [-3000,-1400)
4. [-1400,0)
5. [0, Inf)

- Attribute 6: `AMT_CREDIT`

1. [-Inf,300000)
2. [300000,650000)
3. [650000,900000)
4. [900000,1300000)
5. [1300000, Inf)

- Attribute 7: `DAYS_LAST_PHONE_CHANGE`

1. [-Inf,-2100)
2. [-2100,-1100)
3. [-1100, Inf)

- Attribute 8: `AMT_GOODS_PRICE`

1. [-Inf,150000)
2. [150000,300000)
3. [300000,500000)
4. [500000,700000)
5. [700000,1200000)
6. [1200000, Inf)

- Attribute 9: `EXT_SOURCE_2`

1. [-Inf,0.16)
2. [0.16,0.46)

3. [0.46,0.66)
4. [0.66,0.72)
5. [0.72, Inf)

- Attribute 10: `OCCUPATION_TYPE`
  1. missing
  2. The remaining types
  3. Cleaning staff, Sales staff, Cooking staff, Laborers, Security staff and Waiters/barmen staff
  4. Drivers and Low-skill Laborers
- Attribute 11: `REGION_POPULATION_RELATIVE`
  1. [-Inf,0.02)
  2. [0.02,0.022)
  3. [0.022,0.032)
  4. [0.032,0.036)
  5. [0.036, Inf)
- Attribute 12: `NAME_INCOME_TYPE`
  1. Student and Pensioner
  2. State servant
  3. Commercial associate
  4. Working, Unemployed and Maternity leave
- Attribute 13: `REGION_RATING_CLIENT_W_CITY`
  1. 1
  2. 2
  3. 3
- Attribute 14: `NAME_EDUCATION_TYPE`
  1. Academic degree and Higher education
  2. Incomplete higher, Secondary / secondary special and Lower secondary
- Attribute 15: `REGION_RATING_CLIENT`

1. 1
2. 2
3. 3

- Attribute 16: `AMT_INCOME_TOTAL`
  1. [-Inf,70000)
  2. [70000,220000)
  3. [220000,310000)
  4. [310000, Inf)
- Attribute 17: `CODE_GENDER`
  1. F, N/A
  2. M
- Attribute 18: `REG_CITY_NOT_WORK_CITY`
  1. 0
  2. 1.

3.4 Image transformation

After having constructed the WOE bins for each variable, we converted this information to an image format. For example, considering the first applicant in training data in the first fold of cross-validation performed on the **German Credit data**. This applicant has:

- checking account < 200 DM (Attribute 1 - bin 3)
- age in 35 - 37 (Attribute 2 - bin 4)
- duration > 32 (Attribute 3 - bin 4)
- credit history is no credits taken/all credits paid back duly (Attribute 4 - bin 4)
- saving account is ≥ 500 DM (Attribute 5 - bin 1)
- purpose is radio/television and repairs (Attribute 6 - bin 2)
- property is real estate (Attribute 7 - bin 1)

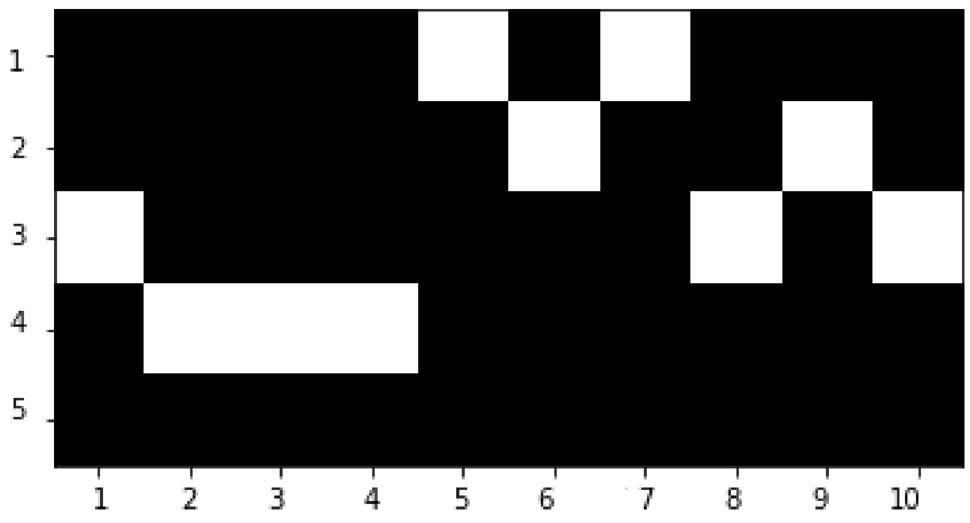


Figure 3.2: The binary image for the first client of the **German Credit data** resulting using the bins for each attribute determined by the WOE technique. Here, the columns 1 - 10 represent the attributes and rows 1 - 5 correspond to the WOE based bins. A pixel is colored white if the attribute value of the client falls into the associated bin.

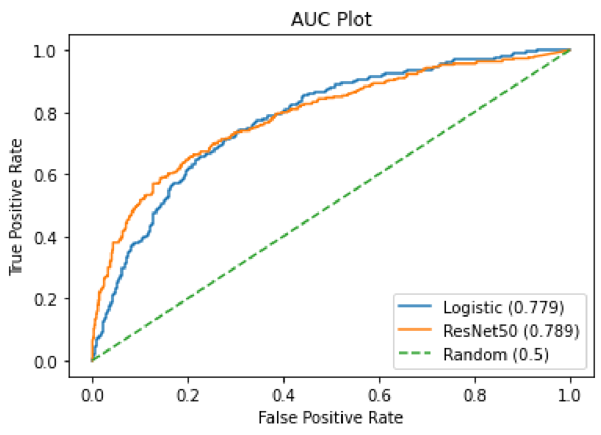
- present employment is 1 and < 4 years (Attribute 8 - bin 3)
- housing is rented (Attribute 9 - bin 2)
- other install is stores (Attribute 10 - bin 3).

The associated binary image is given in Figure 3.2.

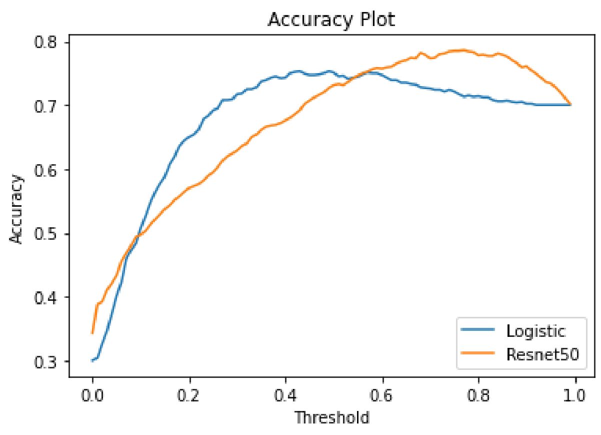
3.5 Credit scoring results

In order to evaluate performances of the ResNet50 and logistic regression for classification, we performed cross-validation with 5 folds. For logistic regression, we performed dummy encoding on categorical variables and kept numeric variables unchanged. This means that all continuous attributes enter linearly into the logistic model. We would like to mention that this might not be the best logistic model to be selected for the data at hand, since nonlinear effects of the attributes and interaction effects are not considered. For the ResNet50, we transformed the transformed information of each client to a binary image. At each fold of cross validation, we trained the ResNet50 and logistic regression on the training data and predicted/evaluated on the test data. The final results were aggregated since each observation is included in the test data of a fold.

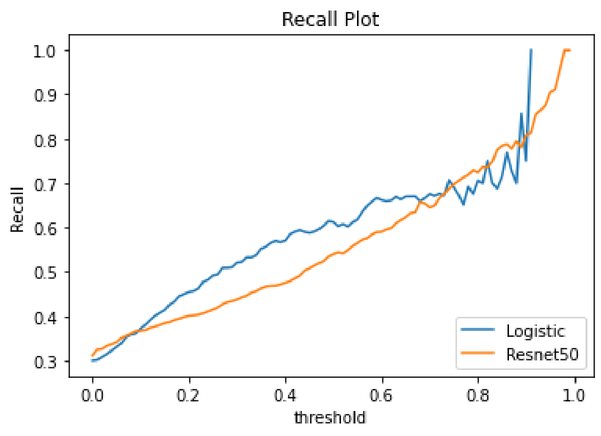
Figure 3.3 represents the comparisons of prediction performances of the ResNet50 and logistic regression on the **German Credit Data** in terms of four metrics discussed before. In terms of Accuracy (Figure 3.3b), logistic regression performs better than the ResNet50 for thresholds from 15% to 60% while the ResNet50 performs better for thresholds > 60%. We see that logistic regression has a higher recall compared to the ResNet50 (Figure 3.3c) while the ResNet50 has a higher precision (Figure 3.3d). In terms of AUC (Figure 3.3a), the ResNet50 performs slightly better than logistic regression. Combining all plots, we can conclude that even though the ResNet50 performs slightly better than logistic regression, logistic regression has a better ability in predicting default cases correctly.



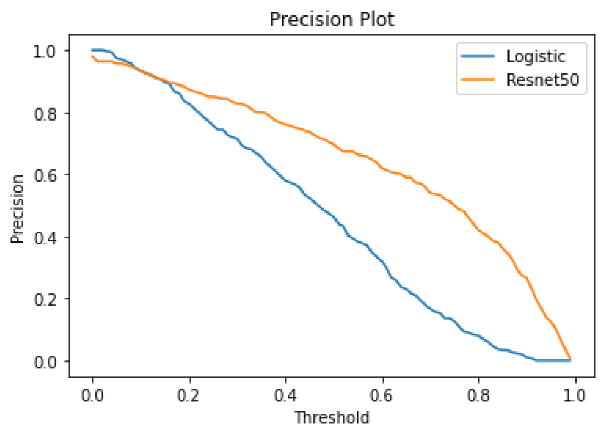
(a) AUC



(b) Accuracy

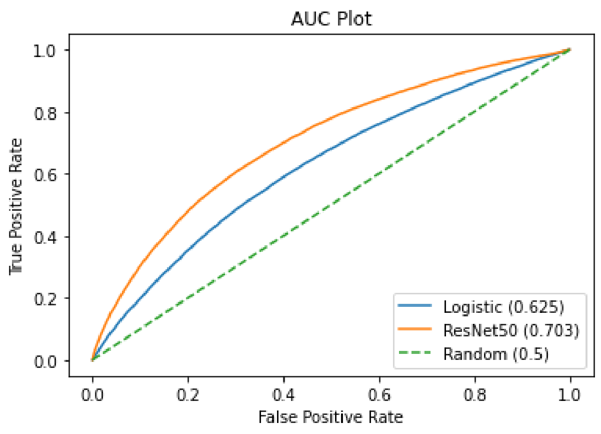


(c) Recall

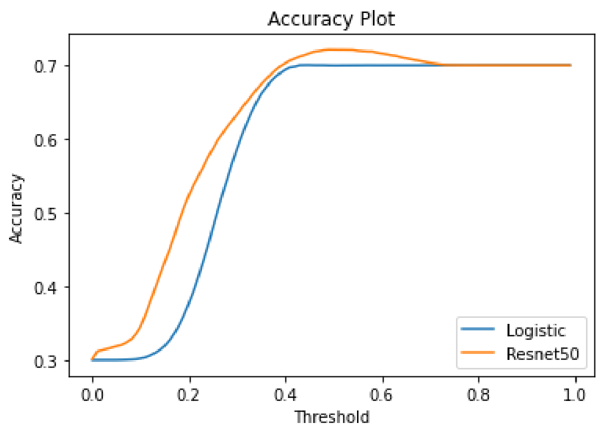


(d) Precision

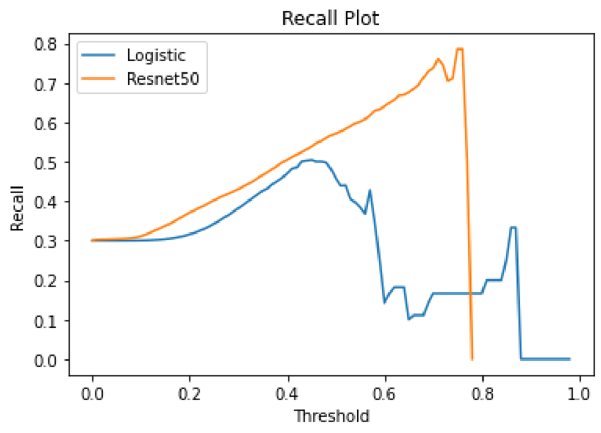
Figure 3.3: Comparison of prediction performances of the ResNet50 and logistic regression on the **German Credit Data**.



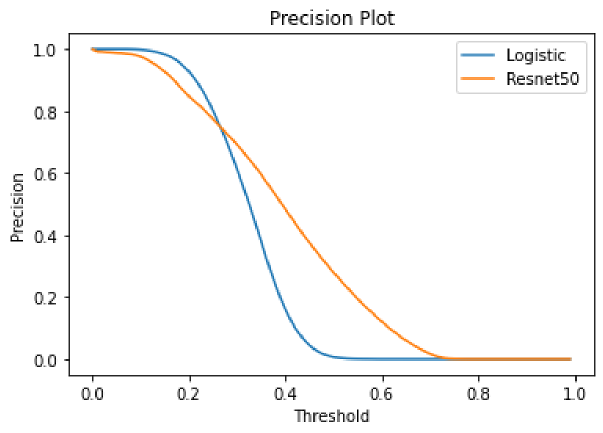
(a) AUC



(b) Accuracy



(c) Recall



(d) Precision

Figure 3.4: Comparison of prediction performances of ResNet50 and logistic regression on **Home Credit Data**.

Now, we consider the much larger data set, which is **Home Credit Data**. Figure 3.4 represents the comparisons of prediction performances of the ResNet50 and logistic regression. It is obvious that the ResNet50 performs better than logistic regression in terms of all criteria. This is what we are expecting since the neural network analysis usually works well on a large data set.

3.6 Shuffle row, column and both of image

When we transform each client's information with the WOE bins to a binary image, there is no rule how to choose the row or column order. Therefore, "How does the model performance change when we change the order of variables or bins? " will be raised. In order to answer this question, we considered 3 scenarios: shuffle in rows, shuffle in columns and shuffle in both rows and columns. We applied these 3 scenarios to both data sets and then retrained the model and re-evaluated the model's performance. Firstly, we considered the first fold of the cross-validation for the **German Credit Data**.

Table 3.3: The order of columns and rows applied on the first fold of the cross-validation on the **German Credit Data**. The row order is 1 - 5 and the column order is 1 - 10.

bin	`1,chk_acct`	`2,age`	`3,duration`	`4,credit_his`	`5,saving_acct`
1	A12,A11	<26	<8	A31,A30	A61
2	A13	[26,35)	[8,16)	A32	A62
3	A14	[35,37)	[16,32)	A33	A64,A63,A65
4		[37,53)	≥ 32	A34	
5		≥ 53			
bin	`6,purpose`	`7,property`	`8,present_emp`	`9,housing`	`10,other_install`
1	A41,A48	A121	A72, A71	A151	A141
2	A410,A40,A46	A122	A73	A152	A142
3	A42	A123	A74	A153	A143
4	A43,A45	A124	A75		
5	A49,A44				

Table 3.3 shows the rules of column and row order applied to the first fold of the cross-validation for the **German Credit Data**. We applied this rule to convert each client's information with the WOE bins to a binary image, called an original image. Then, we randomly shuffled the order of column/row, based on the three mentioned scenarios. The shuffled orders are shown as follows:

- The shuffled row order: 5, 2, 3, 1, 4.
- The shuffled column order: `other_install`, `purpose`, `present_emp`, `duration`, `property`, `chk_acct`, `saving_acct`, `housing`, `credit_his` and `age`.
- The shuffled column and order order: combining the shuffled row order and the shuffled column order.

Table 3.4 presents the AUC results of these 3 scenarios. It is obvious that model performance is almost unchanged. Next, we considered the second fold of the cross-validation for the **German Credit Data**.

Table 3.4: AUC for these 3 scenarios on the first fold of the cross-validation on the **German Credit Data**.

Rule	AUC
images with shuffle in columns	0.795
images with shuffle in rows	0.795
images with shuffle in both columns and rows	0.797
original images	0.798

Table 3.5: The order of columns and rows applied on the second fold of the cross-validation on the **German Credit Data**. The row order is 1 - 6 and the column order is 1 - 10.

bin	1,chk_acct	2,age	3,duration	4,credit_his	5,saving_acct
1	A12,A11	<24	<8	A31,A30	A61
2	A13	[24,26)	[8,14)	A32	A62
3	A14	[26,33)	[14,16)	A33	A64,A63,A65
4		[33,35)	[16,32)	A34	
5		[35,39)	[32,44)		
6		≥ 39	≥ 44		
bin	6,purpose	7,property	8,present_emp	9,housing	10,other_install
1	A43	A121	A72, A71	A151	A141,A142
2	A44-46,A410	A122	A73	A152	A143
3	A48,A41	A123	A74	A153	
4	A4,A45	A124	A75		
5	A49,A42,A40				
6					

Table 3.6: AUC for these 3 scenarios on the second fold of the cross-validation on the **German Credit Data**.

Rule	AUC
images with shuffle in columns	0.8085
images with shuffle in rows	0.8087
images with shuffle in both columns and rows	0.8085
original images	0.808

Table 3.5 shows the rules of column and row order applied to the third fold of the cross-validation for the **German Credit Data**. Then, we randomly shuffled the order of column/row, based on the three mentioned scenarios. The shuffled orders are shown as follows:

- The shuffled row order: 1, 2, 6, 3, 5, 4.
- The shuffled column order: `housing`, `age`, `purpose`, `chk_acct`, `present_emp`, `duration`, `other_install`, `saving_acct`, `credit_his` and `property`.
- The shuffled column and order: combining the shuffled row order and the shuffled column order.

Table 3.6 presents the AUC results of these 3 scenarios. It is obvious that model performance is almost unchanged.

We can conclude that a shuffle in the image column or row has no influence on the model results. This is because convolutional neural networks work by splitting images to many small parts and extracting insights from each part, then aggregating these insights to predict. Hence, no matter how an image is changed, the network still detects similar insights of the image.

3.7 Interpretation of the credit scoring results

For each data set, we randomly chose some images in the test data. Then, we predicted the target class for these randomly selected images. Next, we applied the SHAP technique to identify the contribution of each pixel to the prediction of the response.

Figure 3.5 and Figure 3.7 present some SHAP interpretation for some predictions in the **German Credit Data** and the **Home Credit Data** respectively. The red pixels represent the positive SHAP values that contributed to classifying that image as that particular class, and blue pixels represent the negative SHAP values that contributed to not classifying that image as that particular class. In both figures, the first column is the original image, the second column is the SHAP interpretation for the "Good" label and the last column is the SHAP interpretation for the "Bad" label.

To interpret the prediction in the data context, we focus on the pixels which overlap between the original image and the SHAP interpretation image. These overlaps are used to identify which the pixel contributes more/less to the prediction of a particular class. We focus on the pixels contributing more to the prediction and use the list of WOE bins in Section 3.3.1 to explain these pixels in sentences.

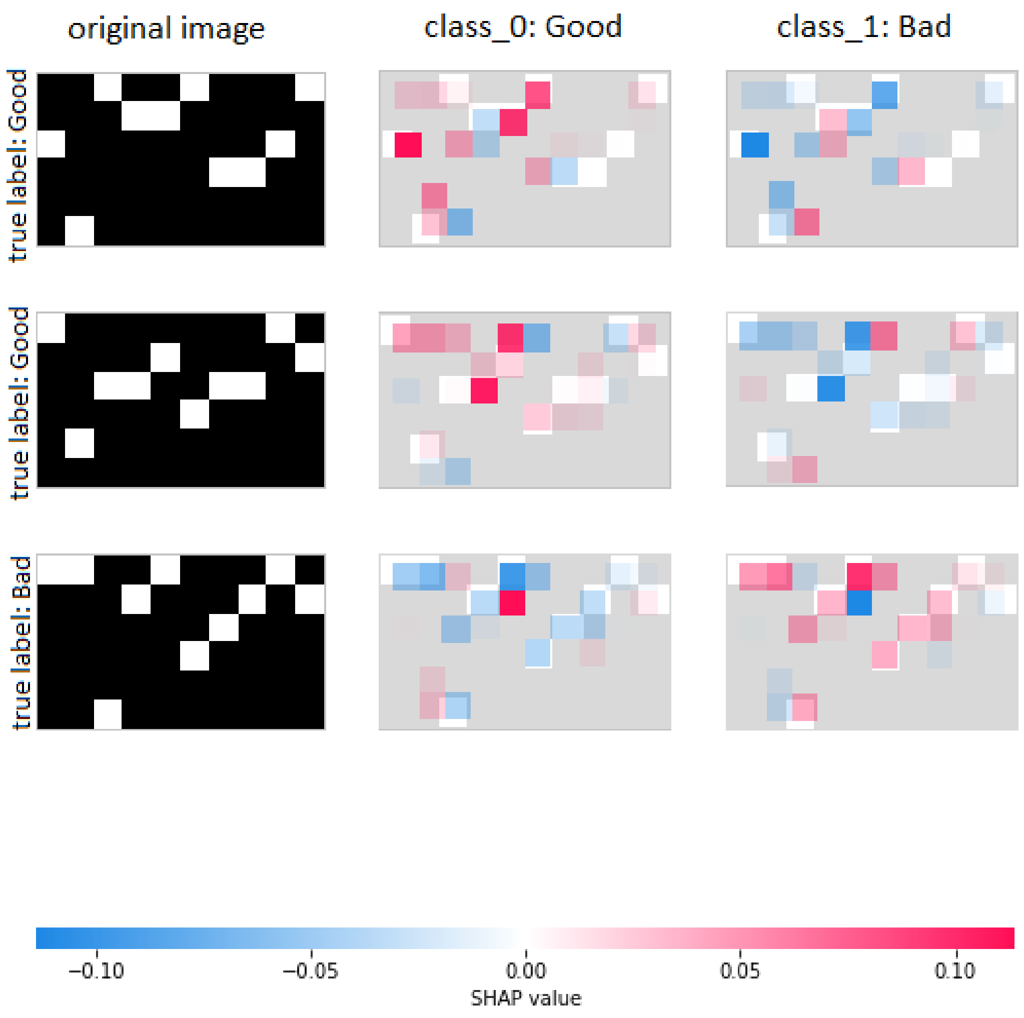


Figure 3.5: The SHAP interpretation for 3 random applicants in the first fold of cross-validation performed on the **German Credit Data**. True label for each original image is good, good and bad respectively.

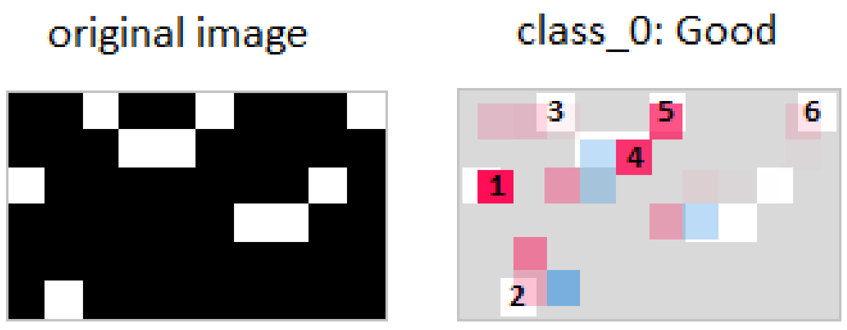


Figure 3.6: An example of the SHAP interpretation for the first applicant in the **German Credit Data**.

For example, we used the first applicant (the first row) in Figure 3.5 and explain why this applicant was classified as “Good”. We looked for pixels which are overlaps between the original image and the SHAP value image for the “Good” label then we selected pixels with the positive SHAP values (red color). As a result (see Figure 3.6), we identified 6 pixels, which can be used to explain why this client is a good client, out of 10 white pixels from the original image. The pixels 1, 4 and 5 identified in Figure 3.6 contribute the most to the prediction for the “Good” label. By looking at the WOE binning list in Section 3.3.1 for these 6 pixels, we can say that the reasons why this applicant should be a good one are:

- Existing checking account is greater than 200 DM (pixel 1 in Figure 3.6)
- Age is greater than or equal 53 (pixel 2 in Figure 3.6)
- Duration is less than 8 months (pixel 3 in Figure 3.6)
- Savings account/bonds is [100, 500 DM) (pixel 4 in Figure 3.6)
- Purpose is car (used) and retraining (pixel 5 in Figure 3.6)
- No other installment plans (pixel 6 in Figure 3.6).

This makes sense in reality since this applicant is of working age, has lots of money in both saving and checking account, and no installment plans and is asking for a short-term loan. Therefore, predicting the “Good” for this applicant seems reasonable.

The Listing 3.1 is the summary of the logistic regression model in the first fold of cross-validation performed on the **German Credit Data**. Based on the column of p-value on the Listing 3.1, `chk_acct`, `duration`, `credit_hist`, `saving_acct` and `property` are statistically significant predictors. These variables also have the highest value for the estimated coefficients, especially `chk_acct` and `saving_acct`. Although the ResNet50 and logistic regression result in different lists of important predictors, the list of two predictors (`chk_acct` and `saving_acct`) with the highest contribution is similar for both methods.

Listing 3.1: The output of logistic regression for the first fold of the **German Credit Data**.

```
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      -3.000293   0.618522  -4.851 1.23e-06 ***
chk_acctA11       1.606089   0.227343   7.065 1.61e-12 ***
chk_acctA12       1.104251   0.227187   4.861 1.17e-06 ***
chk_acctA13       0.641936   0.389731   1.647  0.09953 .
age              -0.007136   0.008778  -0.813  0.41623
duration          0.029680   0.007395   4.013 5.98e-05 ***
credit_hisA30     1.237034   0.448233   2.760  0.00578 **
credit_hisA31     1.397768   0.408390   3.423  0.00062 ***
credit_hisA32     0.612747   0.214386   2.858  0.00426 **
credit_hisA33     0.535870   0.328342   1.632  0.09267 .
saving_acctA61    0.717890   0.246736   2.910  0.00362 **
saving_acctA62    0.627417   0.343375   1.827  0.06767 .
saving_acctA63    0.135612   0.444718   0.305  0.76041
saving_acctA64   -0.532901   0.540305  -0.986  0.32399
propertyA121     -1.045480   0.402939  -2.595  0.00947 **
propertyA122     -0.708346   0.395527  -1.791  0.07331 .
propertyA123     -0.774547   0.387474  -1.999  0.04561 *
present_empA71    0.042881   0.374324   0.115  0.90880
present_empA72    0.293965   0.283115   1.038  0.29912
present_empA73    0.075037   0.246439   0.304  0.76076
present_empA74   -0.638943   0.302595  -2.112  0.03473 *
housingA151       0.766898   0.453344   1.692  0.09071 .
housingA152       0.370612   0.428779   0.864  0.38740
other_installA141  0.635844   0.247090   2.573  0.01007 *
other_installA142  0.492443   0.394455   1.248  0.21188
purposeA40        0.20348    0.29746    0.684  0.49393
purposeA41       -0.76656    0.37736   -2.031  0.04222 *
purposeA410       0.75080    0.68134    1.102  0.27048
purposeA42        0.06023    0.31116    0.194  0.84652
purposeA43       -0.53094    0.30287   -1.753  0.07960 .
purposeA44        0.75080    0.75115    1.000  0.31753
purposeA45        0.43273    0.53161    0.814  0.41564
purposeA46        0.59670    0.40950    1.457  0.14507
purposeA48       -1.03992    1.08933   -0.955  0.33976

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

Now, we give another example using the **Home Credit Data** by considering the first applicant (the first row) in Figure 3.7. Similarly, we identified 10 pixels, which will be used to explain why this client is a bad client, out of 18 white pixels from the original image (see Figure 3.8). The pixels 7 and 9 contributed the most to the prediction. By looking at the WOE binning list in Section 3.3.1 for these 10 pixels, we can say that the reasons why this applicant should be a bad one are:

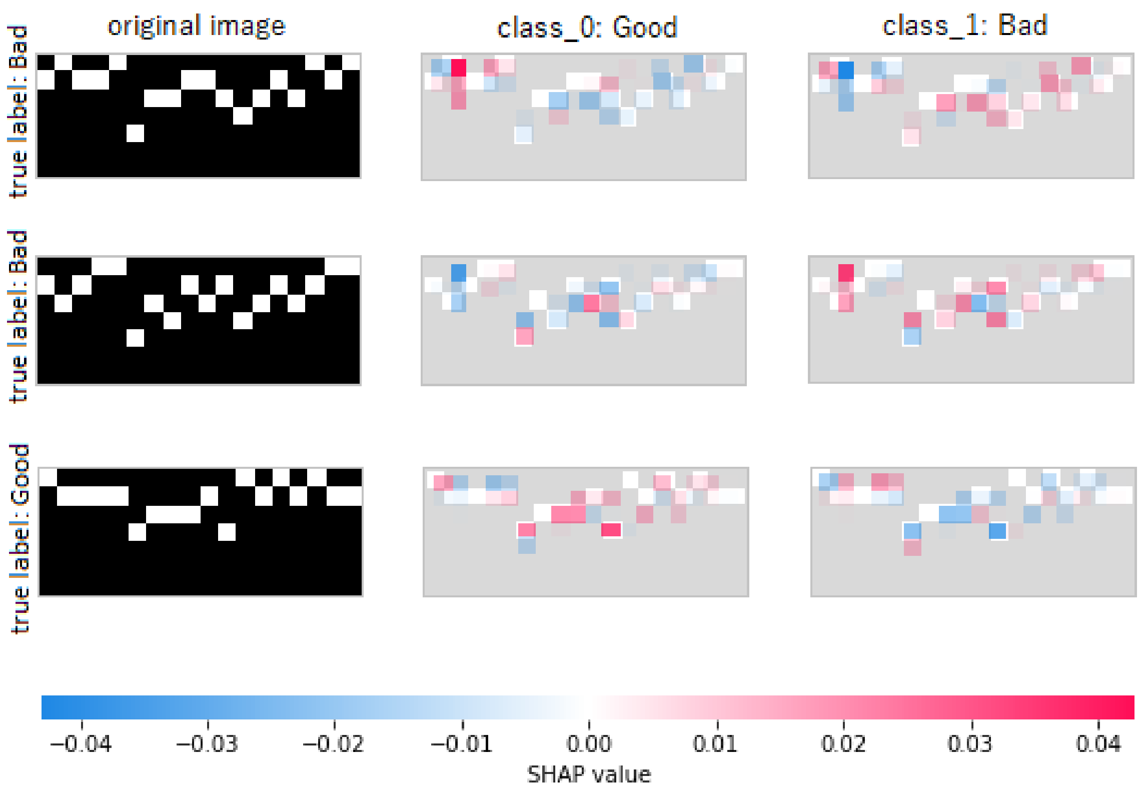


Figure 3.7: The SHAP interpretation on 3 random applicants in the fifth fold of cross-validation performed on the **Home Credit Data**. The true label for each original image is bad, bad and good respectively.

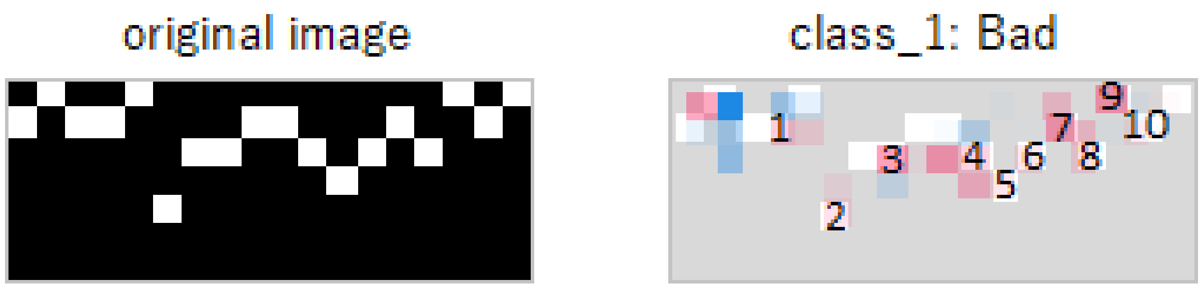


Figure 3.8: An example of the SHAP interpretation for the first applicant in the **Home Credit Data**.

- DAYS_ID_PUBLISH is [-4200,-3200) (pixel 1 in Figure 3.8)
- AMT_CREDIT is [1300000, Inf) (pixel 2 in Figure 3.8)
- AMT_GOODS_PRICE is [300000,500000) (pixel 3 in Figure 3.8)
- REGION_POPULATION_RELATIVE is [0.022,0.032) (pixel 4 in Figure 3.8)
- NAME_INCOME_TYPE is working, unemployed and maternity leave (pixel 5 in Figure 3.8)
- REGION_RATING_CLIENT_W_CITY is 3 (pixel 6 in Figure 3.8)
- NAME_EDUCATION_TYPE is incomplete higher, secondary / secondary special and lower secondary (pixel 7 in Figure 3.8)
- REGION_RATING_CLIENT is 3 (pixel 8 in Figure 3.8)
- AMT_INCOME_TOTAL is [-Inf,70000) (pixel 9 in Figure 3.8)
- CODE_GENDER is M (pixel 10 in Figure 3.8).

This interpretation is reasonable since this applicant has a low level of education, income and annuity, and has the worst rating of living place but is applying for a large loan. Therefore, the "Bad" label should be tagged for this applicant.

The Listing 3.2 is the summary of logistic regression model in the fifth fold of cross-validation performed on the **Home Credit Data**. Based on the column of p-value on the Listing 3.2, REGION_POPULATION_RELATIVE, OCCUPATION_TYPE, NAME_EDUCATION_TYPE and AMT_INCOME_TOTAL are not statistically significant predictors. Among statistically significant predictors, EXT_SOURCE_2 has the highest contribution to the prediction of the response. Since there is a big difference in the two methods' performance, we expect the lists of important predictors to be different as well.

Listing 3.2: The output of logistic regression for the first fold of the **Home Credit Data**.

```
                                                Estimate Std. Error z value Pr(>|z|)
(Intercept)                                    1.495e+02  1.240e+01  12.051  < 2e-16 ***
DAYS_BIRTH                                     2.320e-05  2.960e-06   7.838 4.57e-15 ***
AMT_ANNUITY                                    9.757e-06  1.037e-06   9.411  < 2e-16 ***
DAYS_REGISTRATION                              1.650e-05  2.828e-06   5.835 5.37e-09 ***
DAYS_ID_PUBLISH                                6.671e-05  6.243e-06  10.684  < 2e-16 ***
DAYS_EMPLOYED                                  8.221e-05  5.229e-06  15.722  < 2e-16 ***
AMT_CREDIT                                     2.783e-06  1.421e-07  19.589  < 2e-16 ***
DAYS_LAST_PHONE_CHANGE                         7.236e-05  1.180e-05   6.131 8.75e-10 ***
AMT_GOODS_PRICE                               -3.390e-06  1.597e-07 -21.222  < 2e-16 ***
EXT_SOURCE_2                                  -2.267e+00  4.734e-02 -47.884  < 2e-16 ***
```

```
REGION_POPULATION_RELATIVE                         7.238e-01  8.194e-01   0.883  0.37709    
OCCUPATION_TYPE                                   -2.000e-01  1.290e-01  -1.550  0.12115    
OCCUPATION_TYPEAccountants                        -4.454e-01  1.398e-01  -3.186  0.00144 ** 
OCCUPATION_TYPECleaning.staff                     -9.096e-02  1.439e-01  -0.632  0.52745    
OCCUPATION_TYPECooking.staff                      -1.061e-01  1.393e-01  -0.761  0.44637    
OCCUPATION_TYPECore.staff                         -3.114e-01  1.310e-01  -2.378  0.01741 *  
OCCUPATION_TYPEDrivers                            -5.867e-02  1.321e-01  -0.444  0.65705    
OCCUPATION_TYPEHigh.skill.tech.staff              -3.191e-01  1.367e-01  -2.335  0.01956 *  
OCCUPATION_TYPEHR.staff                           -2.204e-01  2.621e-01  -0.841  0.40040    
OCCUPATION_TYPEIT.staff                           -3.545e-01  2.704e-01  -1.311  0.18984    
OCCUPATION_TYPELaborers                           -9.725e-02  1.285e-01  -0.757  0.44912    
OCCUPATION_TYPELow.skill.Laborers                  8.276e-02  1.542e-01   0.537  0.59136    
OCCUPATION_TYPEManagers                           -2.540e-01  1.327e-01  -1.914  0.05558 .  
OCCUPATION_TYPEMedicine.staff                     -2.933e-01  1.395e-01  -2.103  0.03549 *  
OCCUPATION_TYPEPrivate.service.staff              -3.983e-01  1.636e-01  -2.434  0.01492 *  
OCCUPATION_TYPERealty.agents                      -2.181e-02  2.186e-01  -0.100  0.92055    
OCCUPATION_TYPESales.staff                        -1.339e-01  1.291e-01  -1.037  0.29994    
OCCUPATION_TYPESecretaries                        -2.457e-01  1.953e-01  -1.258  0.20837    
OCCUPATION_TYPESecurity.staff                     -9.916e-02  1.393e-01  -0.712  0.47659    
NAME_INCOME_TYPEBusinessman                       -3.823e+00  2.332e+00  -1.639  0.10116    
NAME_INCOME_TYPECommercial.associate              -1.093e+00  1.022e+00  -1.069  0.28514    
NAME_INCOME_TYPEPensioner                         -1.789e+02  1.255e+01 -14.257  < 2e-16 ***
NAME_INCOME_TYPEState.servant                     -1.214e+00  1.023e+00  -1.187  0.23541    
NAME_INCOME_TYPEUnemployed                        -1.775e+02  1.256e+01 -14.132  < 2e-16 ***
NAME_INCOME_TYPEWorking                           -9.686e-01  1.022e+00  -0.948  0.34335    
REGION_RATING_CLIENT_W_CITY                        3.030e-01  6.171e-02   4.910 9.12e-07 ***
REGION_RATING_CLIENT                              -1.266e-01  6.103e-02  -2.075  0.03802 *  
NAME_EDUCATION_TYPEAcademic.degree                -1.481e+00  5.754e-01  -2.573  0.01008 *  
NAME_EDUCATION_TYPEHigher.education               -3.482e-01  2.500e-02 -13.928  < 2e-16 ***
NAME_EDUCATION_TYPEIncomplete.higher              -1.448e-01  5.013e-02  -2.888  0.00388 ** 
NAME_EDUCATION_TYPELower.secondary                 7.014e-02  7.479e-02   0.938  0.34828    
AMT_INCOME_TOTAL                                   3.784e-08  3.384e-08   1.118  0.26348    
CODE_GENDERF                                      -2.068e-01  2.205e-02  -9.379  < 2e-16 ***
REG_CITY_NOT_WORK_CITY                             8.700e-02  2.131e-02   4.082 4.46e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

Chapter 4

Conclusion

In this thesis, we propose a novel method which utilizes residual neural networks for credit scoring. The proposed approach converts tabular data sets into images and thus allows the application of a residual neural network with 50 layers (ResNet50) in credit scoring. Each pixel of the image corresponds to a feature bin of the tabular data set. We did the experiments for the proposed method and logistic regression for the **German Credit Data** and the **Home Credit Data**. Based on the results, our proposed approach shows superiority to logistic regression, especially, if the sample size is large. We also performed some interpretation for the ResNet's prediction by using the SHAP method. It estimates the contribution of each pixel to the prediction of the response. In this way, we can identify which pixel has a positive/negative contribution to the prediction of the response depending on the sign of the Shapley value of the pixel. To interpret them in the data context, all important pixels were linked to their respective bins that were obtained by the WOE transformation.

We believe that neural networks in computer vision show the great promise for the credit scoring industry since they show good results and allow for nice visualization. There are several possible extensions to improve its performance in the credit scoring industry:

- Apply feature engineering methods to make predictors more predictive (Jeff [2017]).
- Apply feature selection methods to reduce the number of predictors (Isabelle and Andre [2003]).
- Apply the GAN-based augmentation or adversarial training to make training data less imbalanced (Goodfellow [2016]).
- Apply other deep neural networks such as long short-term memory neural networks or deep graph neural networks (Sepp and Jürgen [1997]).

List of Figures

List of Tables

Bibliography

Zakrzewska, D. (2007). *On integrating unsupervised and supervised classification for credit risk evaluation*. Information Technology and Control, 36(1), 98-102.

Thomas, L.C. (2000). *A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers*. International Journal of Forecasting, 16(2):149–172.

Abrahams, R. & Zhang, M. (2008). *Fair lending compliance: Intelligence and implications for credit risk management*. John Wiley & Sons.

Akhavein, D. & Frame, W. & White, J. (2005). *The Diffusion of Financial Innovations: An Examination of the Adoption of Small Business Credit Scoring by Large Banking Organizations*. Journal of Business, Vol 78, No. 2.

Thomas, L.C. & Edelman, D.B. & Crook, J.N. (2002). *Credit Scoring and its Applications*. SIAM, Philadelphia.

Meyer, P. & Pifer, H. (1970). *Prediction of Bank Failures*. Journal of Finance (September 1970).

Kass, G. V. (1980). *An Exploratory Technique for Investigating Large Quantities of Categorical Data*. Journal of the Royal Statistical Society. Series C (Applied Statistics), vol. 29, no.2, 1980, pp. 119-27.

Pearson, K. (1900). *On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling*. Philosophical Magazine. Series 5. 50 (302): 157-17.

Siddiqi, N. (2005). *Credit Risk Scorecards: Developing And Implementing Intelligent Credit Scoring*. Chennai, India: SAS, pp. 78–79.

King, G. (1989). *Unifying Political Methodology: The Likelihood Theory of Statistical Inference*. New York: Cambridge University Press. p. 84. ISBN 0-521-36697-6.

Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems, 2(4):303–314.

Hornik, K. (1991). *Approximation capabilities of multilayer feedforward networks*. Neural Networks, 4(2):251–257.

LeCun, Y. (1986). *Learning Processes in an Asymmetric Threshold Network*, in Bienenstock, E. and Fogelman-Soulié, F. and Weisbuch, G. (Eds), Disordered systems and biological organization, 233-240, Springer-Verlag, Les Houches, France.

Polyak, B.T. (1964). *Some methods of speeding up the convergence of iteration methods*. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17.

Nesterov, Y. (1983). *A method of solving a complex programming problem with convergence rate o(1/k2)*. Soviet Mathematics Doklady, 27:372– 376.

Sutskever, I. & Martens, J. & Dahl, G.E. & Hinton, G.E. (2013). *On the importance of initialization and momentum in deep learning*. ICML, 28(3):1139–1147.

LeCun, Y. & Jackel, L. & Boser, B. & Denker, J. & Graf, H. & Guyon, I. & Henderson, D. & Howard, R. & Hubbard, W. (1998). *Handwritten digit recognition: Applications of neural networks chipsand automatic learning*. Proceedings of the IEEE, 86(11):2278–2324.

Zhifei, Z. (2016). *Derivation of Backpropagation in Convolutional Neural Network (CNN).*

Szegedy, C. & Ioffe, S. & Vanhouche, V. & Alemi, A. (2016). *Inception-v4, inception-resnet and the impact of residual connections on learning*. Arxiv, 1602.07261.

Kaiming, H. & Xiangyu, Z. & Shaoqing, R. & Jian, S. (2015). *Deep Residual Learning for Image Recognition*. CoRR abs/1512.03385.

Rumelhart, D.E. & Hinton, G.E. & Williams, R.J., et al. (1988). *Learning Representations by Back-Propagating Errors*. Cognitive Modeling, 5, 1.

Glorot, X. & Bengio, Y. (2010). *Understanding the Difficulty of Training Deep Feedforward Neural Networks*. In: Teh, Y.W. and Titterington, M., Eds., Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research (PMLR), Chia Laguna Resort, Sardinia, Italy, Vol. 9.

Hans, H. (1995). UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data).

Kaggle. (2018). Home Credit Default Risk. [Online]. Available: https://www.kaggle.com/c/home-credit-default-risk.

Breiman, L. (1984). *Classification and regression trees*. The Wadsworth and Brooks-Cole statisticsprobability series. Chapman & Hall.

Scott, L. & Lee, S. (2017). *A Unified Approach to Interpreting Model Predictions*, CoRR, vol. abs/1705.07874, pp. 1–10.

Shapley, L. S. (1953). *A Value for N-Person Games*. Contributions to the Theory of Games 2, 307–317.

Strumbel, E. & Kononenko, I. (2010). *An Efficient Explanation of Individual Classifications using Game Theory*. Journal of Machine Learning Research 11, 1–18.

Strumbel, E. & Kononenko, I. (2014). *Explaining prediction models and individual predictions with feature contributions*. Knowledge and Information Systems 41, 647–665.

Kjersti, A. & Martin, J. & Anders, L. (2019). *Explaining individual predictions when features are dependent: More accurate approximations to Shapley values*.

Huang, C.L. & Chen, C.H. & Wang C.J. (2007). *Credit scoring with a data mining approach based on support vectormachines*. Expert Systems with Applications, 33, pp. 847-856.

Huang, Z. & Chen, H. & Hsu, C. & Chen, W. & Wu, S. (2004). *Credit rating analysis with support vector machines and neural networks: A market comparative study*. Decision Support Systems. 37. 543-558.

Ince, H. & Aktan, B. (2009). *A comparison of data mining techniques for credit scoring in banking: A managerial perspective*. Journal of Business Economics and Management - J BUS ECON MANAG. 10. 233-240.

Provenzano, A. R. & Trifirò, D. & Datteo, A. & Giada, L. & Jean, N. & Riciputi, A. & Le Pera, G. & Spadaccino, M. & Massaron, L. & Nordio, C. (2020). *Machine Learning approach for Credit Scoring*, arXiv.

Liu, C. & Huang, H. & Lu, S. (2019). *Research on Personal Credit Scoring Model Based on Artificial Intelligence*. 10.1007/978-3-030-15740-1-64.

Sharma, D. (2011). *Improving the Art, Craft and Science of Economic Credit Risk Scorecards Using Random Forests: Why Credit Scorers and Economists Should Use Random Forests*. Available at SSRN: https://ssrn.com/abstract=1861535.

Aida, K.A. (2015). *Bank Credit Risk Analysis with K-Nearest-Neighbor Classifier: Case of Tunisian Banks*. Journal of Accounting and Management Information Systems, Faculty of Accounting and Management Information Systems, The Bucharest University of Economic Studies, vol. 14(1), pages 79-106.

Roger, B. (1991). *Game Theory: Analysis of Conflict, Harvard University Press*, p. 1. Chapter-preview links, pp. vii–xi.

Vijay, S.D. & Jonathan, N.C. & George, A.O. (1996). *A comparison of neural networks and linear scoring models in the credit union environment*. European Journal of Operational Research, vol. 95, issue 1, 24-37.

Goodfellow, I. (2016). *Nips 2016 tutorial: Generative adversarial networks*. arXiv preprint arXiv: 1701.00160.

Sepp, H. & Jürgen, S. (1997). *Long short-term memory*. Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276. S2CID 1915014.

Jeff, H. (1997). *An Empirical Analysis of Feature Engineering for Predictive Modeling*. arXiv:1701.07852v2.

Isabelle, G. & Andre, E. (2003). *An Introduction to Variable and Feature Selection*. Journal of Machine Learning Research 3 (2003) 1157-1182.