Kvalificēts elektroniskais paraksts
OLEKSII AVDIEIEV
2022-05-28 22:48:51 GMT+3
Mērķis: Paraksts

RIGA TECHNICAL UNIVERSITY
Faculty of Information Technology and Computer Science
Institute of Applied Computer Systems
Department of Artificial Intelligence and System Engineering

Oleksii Avdieiev
Academic Bachelor Study Program “Computer Systems”
Student ID 191ADB057

COMPARISON OF LOSS FUNCTIONS FOR IMAGE CLASSIFICATION IN THE DEEP LEARNING

BACHELOR PAPER

Scientific adviser
Doctor of Science, Researcher
ĒVALDS URTĀNS

RIGA 2022

RIGA TECHNICAL UNIVERSITY
Faculty of Information Technology and Computer Science
Institute of Applied Computer Systems
Department of Artificial Intelligence and System Engineering

Work Performance and Assessment Sheet of the Bachelor Paper

The author of the graduation paper:

Student Oleksii Avdieiev _____

(signature, date)

The graduation paper has been approved for the defense:

Scientific adviser:

Doctor of Science, Researcher, Ēvalds Urtāns _____

(signature, date)

ABSTRACT

DEEP LEARNING, LOSS FUNCTION, IMAGE CLASSIFICATION, NEURAL NETWORK, COMPARISON

This paper focuses on the comparison of loss functions in different configurations for three datasets: Fashion MNIST, CIFAR-10 and KMNIST. Loss functions performance is compared as they are tested with different probability distribution functions. Four loss functions are compared in this paper, Categorical Cross Entropy loss, KL divergence, JS divergence and Focal loss (two configurations). Three probability distribution functions, Softmax, Soft-margin Softmax and Taylor Softmax are used in experiments with these loss functions. Results have shown that configurations with Categorical Cross Entropy loss produced results with best accuracy. While configurations with Focal with gamma equal to 0.5 converged fastest.

The bachelor thesis contains 67 pages, 41 figure and 36 reference sources.

ANOTĀCIJA

DZIĻA MĀCĪŠANĀS, ZUDUMA FUNKCIJA, ATTĒLU KLASIFIKĀCIJA, NEIRONU TĪKLS, SALĪDZINĀJUMS

Bakalaura darbs ir fokusēts zudumu funkciju salīdzināšanai dažādās konfigurācijās trim datu kopām: Fashion MNIST, CIFAR-10 un KMNIST. Zaudējumu funkciju veiktspēja tiek salīdzināta, kad tās tiek pārbaudītas ar dažādām varbūtības sadalījuma funkcijām. Darbā tiek salīdzinātas četras zudumu funkcijas: Kategoriskais šķērsvirziena entropy zudums, KL nobīde, JS nobīde un fokusa zudums (divas konfigurācijas). Ar šīm zuduma funkcijām eksperimentos izmanto trīs varbūtības sadalījuma funkcijas – Softmax, Soft-margin Softmax un Taylor Softmax. Rezultāti liecina, ka konfigurācijas ar kategorijas šķērsvirziena entropy zudumu nodrošina rezultātus ar vislabāko precizitāti. Kaut arī konfigurācijas ar fokusu ar gamma, kas vienāda ar 0,5, ir visātrākās.

Bakalaura darbs sastāv no 67 lappusēm, 41 attēla un 36 uzziņu avotiem.

INTRODUCTION

Deep learning (DL) is a rapidly developing subfield of machine learning. Even though the experiments with deep neural networks (DNN) started in 1970s, it wasn't until 2011 that DL methods captured the interest of academics and entrepreneurs. This started with successful application of DNNs in academic image-classification competitions, however the real turning point occurred in 2012 because of ImageNet classification challenge. In ImageNet challenge deep convolutional neural network was used and the result was 9.3% increase in accuracy over best model of 2011 (Chollet, 2018; Russell and Norvig, 2021). The importance of this event in development of DL cannot be overstated, according to AI index reports produced by Stanford following major changes can be highlighted (Yoav Shoham *et al.*, 2018; Raymond Perrault *et al.*, 2019; Daniel Zhang *et al.*, 2022):

1) in period from 2010 to 2019 the number of publications about AI has increased 20-fold;
2) the number of students pursuing AI specialization increased by 16 times internationally in comparison to 2010 value;
3) performance in image classification is higher, costs less, moreover the accuracy in object detection exceeds human-level performance;
4) question answering accuracy on Stanford Question Answering dataset exceeds human-level performance;
5) investment into AI is at all-time high totaling $93.5 billion in 2021.

This research is aimed at determining the most suitable loss function for image classification by comparing deep learning model performance with different loss functions and hyper-parameters. To reach this goal following tasks should be completed:

1) relevant literature collection and review;
2) development of the methodology for the experimental part of the research which involves choice of following elements:
   - deep learning model,
   - optimizer,
   - datasets,
   - probability distribution functions,
   - loss functions,

- benchmarks.

3) development of the code for experimenting, this part is performed utilizing the capabilities of PyTorch framework with TensorBoard for results documenting;
4) perform experiments using HPC;
5) analyze obtained results and make conclusions with regards to used loss functions and probability distribution functions;
6) determine possibilities for further research and create an outline.

Work is split into 5 sections.

First chapter encompasses basic concepts of deep learning applicable to most tasks in the field, it delves into building blocks needed to create any model. It goes into shallow explanation of layers, loss functions, backpropagation, optimizers, and basic metrics.

Second chapter focuses on classification tasks, specifically image classification. It explores inputs and their processing, example architecture for classification and probability distribution functions.

Third chapter is about methodology that will be used for experiments, it is important to determine the network architecture, loss functions that will be compared, probability distribution functions to get predictions for specific classes and describe datasets that are going to be used.

Fourth chapter is based on comparison of experimental results and conclusions.

Fifth chapter is the outline for possibilities in further research regarding loss functions in image classification tasks.

RELEVANCE OF WORK

Currently there is a lot of research pertaining loss functions. The number of results from using search term “loss function” and applying filters to narrow the results to those in artificial intelligence field is at all times high and still rising. Three different sources for scientific literatures were used. IEEE Xplore, Semantic Scholar, and Springer Link. The search term loss function was used in all three sources.

For IEEE Xplore, the applied filters were in a “Publication Topics” category, following topics were chosen:

1) learning (artificial intelligence),
2) neural nets,
3) convolutional neural nets,
4) image classification,
5) deep learning (artificial intelligence).

For Springer Link, the applied filter is in the “Subdiscipline” category, and it is artificial intelligence.

For Semantic Scholar, the applied filters are in three different categories and several subcategories:

1) “Computer Science” subcategory in “Fields of Study” category;
2) “Journal Article” subcategory in “Publication Type” category;
3) in “Journals & Conferences” category, following subcategories are chosen:
   - “PLOS One”,
   - “ArXiv”,
   - “Proceedings of the National Academy of Sciences”,
   - “Scientific reports”.

Table 1.1. illustrates the number of search results.

Table 1.1

Number of search results for “loss function”

Year	Semantic Scholar	IEEE Xplore	Springer Link
2012	671	55	3954
2013	830	65	3826
2014	919	55	3748

Year	Semantic Scholar	IEEE Xplore	Springer Link
2015	1183	107	4048
2016	1414	130	5011
2017	1679	306	5841
2018	2351	586	8858
2019	3289	1164	10962
2020	4619	1407	11216
2021	6608	1604	15366

The data in this table indicates that the research of loss functions is relevant to scientific community.

This work focuses on comparison of loss functions for image classification with different datasets and probability distribution functions. Some similar works in this direction are:

1) A Performance Comparison of Loss Functions (Cho *et al.*, 2019),
2) Comparison of Loss Functions for Training of Deep Neural Networks in Shogi (Zhu and Kaneko, 2018),
3) A Performance Comparison of Loss Functions for Deep Face Recognition (Srivastava, Murali and Dubey, 2019).

The difference in all these works is the task under consideration as well as the experimental configuration.

1. DEEP LEARNING FUNDAMENTALS

This research revolves around the topic of image-classification using deep learning methods. Therefore, fundamental concepts of deep learning must be explained.

1.1. Architectures of ANN

Deep learning is a subset of machine learning (ML) thus it inherits its elements. Ideas behind DL are similar to ML, as both fields incorporate learning from example. Learning from example means that ML algorithm is given the data to be processed and outputs associated with this data. These inputs are used to create a set of rules by which data abides. The rules are determined by finding the correlations between features of the data and its related output. The rules can be used with new data to produce answers. This process can be interpreted as mapping of inputs to data by analyzing interdependence of inputs (Buduma and Locascio, 2017; Chollet, 2018). In the book (Chollet, 2018), learning is described as: "Learning, in the context of machine learning, describes an automatic search process for better representations." Therefore, the central task of machine learning is to transform data in such way to get representations that lead to expected output. Representations in this context are transformations of data (Chollet, 2018).

This learning process is performed by training a neural network (NN). Neural networks consist of neurons that are interconnected to emulate the human brain capabilities to transform stimulus to a reaction. Inputs to outputs in artificial representation of brain which is neural network (Buduma and Locascio, 2017). Neurons in case of neural networks are mathematical functions which transform input data. To learn, the neurons are organized into layers which are interconnected. Each neural network consists of input layer, output layer and number of hidden layers which is determined by the developer of neural network. Simplest neural network consists of input and output layer, without any hidden layers (Vasilev *et al.*, 2019). To train neural network following objects are needed (Chollet, 2018):

1) layers which are combined into a network;
2) inputs with corresponding target data;
3) loss function, which is used for learning;
4) optimizer, which is needed to determine how the learning process will

proceed.

This structure is illustrated in Figure 1.1, loss function and optimizer will be discussed in more details later in this chapter.

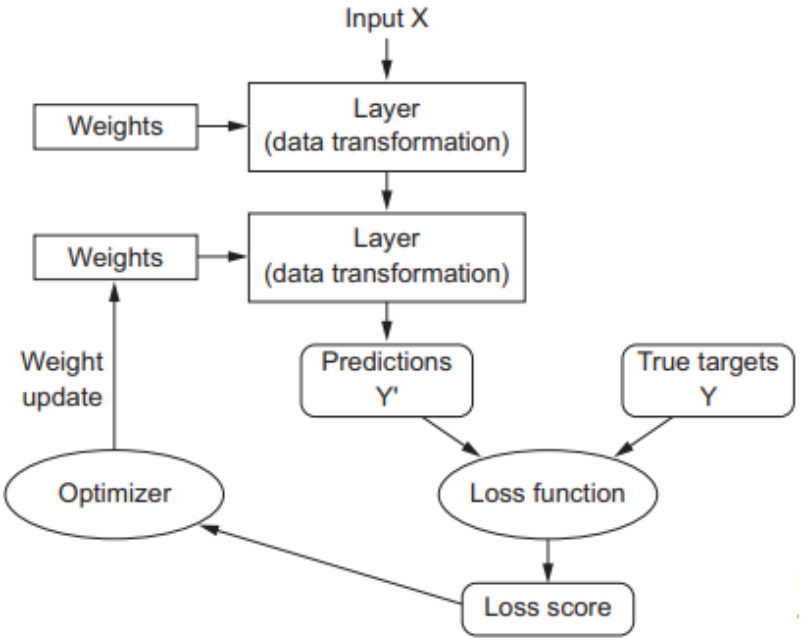

Fig. 1.1. Simple neural network structure (adopted from (Chollet, 2018))

Layers in neural network must be compatible, which means that number and shape of outputs of a layer should be identical to number of inputs and shape of a following layer. Otherwise, it will be impossible to transfer data from one layer to another.

The difference between DL and ML is observed in the architecture of neural networks, that consist of more subsequent layers. This allows to transform and capture data in a more complex way and optimizes the learning process in comparison to single layer networks which depend on linear function to classify data (Chollet, 2018; Russell and Norvig, 2021). Deep learning networks distill information via passing it through numerous layers of transformations and purifying it (Chollet, 2018).

Neural network can consist of different types of layers, the simplest one is linear layer. Linear layer is characterized by equation of a line Formula (1.1).

$$y = W \cdot x + b, \tag{1.1}$$

where y – output of the layer;

W – weight;

x – input to the layer;

b – bias.

Weights in Formula (1.1) are values that are used to transform data in layers. For that reason, learning revolves around finding such values for weights that will produce correct outputs when applied to inputs. Bias can be interpreted as a constant that is applied to shift the output, it needs to be used so the model is not fixed to inputs and can move during training process (Aggarwal, 2018; Chollet, 2018).

There are several reasons why models can not consist only of linear layers:

1) stacked linear layers can be mathematically expressed as one linear layer as seen in Formulas (1.2; 1.3) (Buduma and Locascio, 2017);

$$y_0 = W_0 \cdot x + b_0, \tag{1.2}$$

where y_0 – output of first linear layer;

W_0 – weight of first linear layer;

x – input of first linear layer;

b_0 – bias of first linear layer.

$$y_1 = W_1 \cdot y_0 + b_1 = W_1 \cdot (W_0 \cdot x + b_0) + b_1 = (W_1 \cdot W_0 \cdot x) + (W_1 \cdot b_0 + b_1) = W \cdot x + b, \tag{1.3}$$

where y_1 – output of second linear layer;

W_1 – weight of second linear layer;

y_0 – output of first linear layer;

b_1 – bias of second linear layer;

y_0 – output of first linear layer;

W_0 – weight of first linear layer;

x_0 – input of first linear layer;

b_0 – bias of first linear layer;

W – simplified dot product of W_1 and W_0;

b – simplified addition of biases.

2) linear layers can produce only linearities.

As a consequence of these limitations, it is impossible to learn non-linear transformations data with models restricted to linear layers. This leads to the need of activation functions which are non-linear and allow non-linear transformations (Chollet, 2018).

To determine if the function is an activation function several rules should be followed (Trask, 2019):

1) function must be continuous and infinite in domain (otherwise a problem may occur when input does not have an output);
2) function should be monotonic (otherwise one input can may produce several outputs);
3) in most cases function should be non-linear (to create a more flexible model, which can better adapt to different data);
4) function and its derivative should be efficiently computable.

The simplest activation function is an identity function which is represented by Formula (1.4). The range of values for the function is $(-\infty, \infty)$.

$$f(x) = x, \tag{1.4}$$

where x is input to function.

This function is linear which is seen in Figure 1.2.

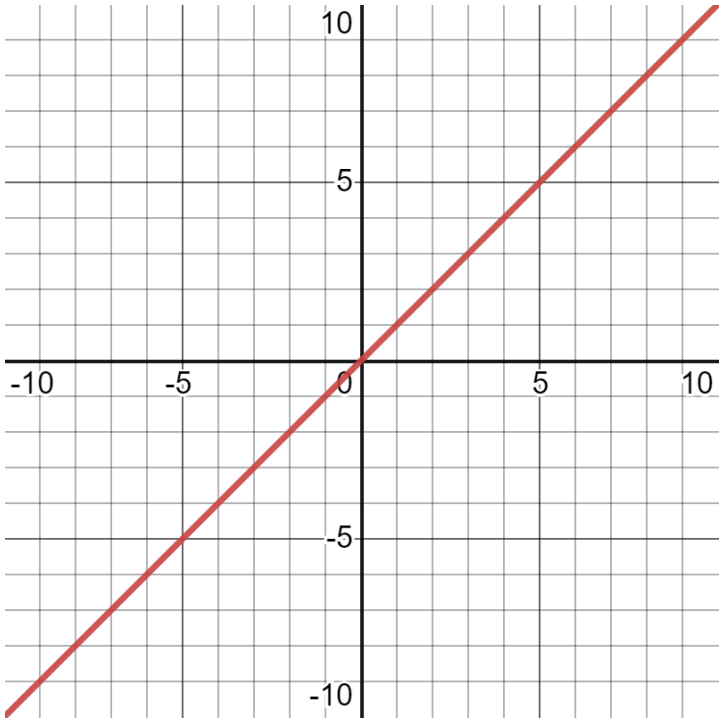

Fig. 1.2. Plot of identity function

To reach non-linearity there is a magnitude of functions that could be used, some widely used examples include (Vasilev *et al.*, 2019):

1) sigmoid,

2) hyperbolic tangent (tanh),
3) rectified linear unit (ReLu),
4) softmax.

To illustrate non-linearity sigmoid, tanh and ReLu will be used. Softmax will be overviewed later in subchapter 2.3 as their results can be interpreted as probability distributions.

Sigmoid serves as a non-linear layer which squishes its input to range $(0, 1)$. This may be helpful in binary classification tasks because the output can be treated as a probability. It is used not only as a hidden layer, but in some cases as an output layer. With appearance of more efficient alternatives sigmoid lost its popularity, however it is still quite useful for binary classification (Aggarwal, 2018; Trask, 2019). Formula (1.5) represents sigmoid.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.5}$$

where x – input to function;

e – exponent.

Plot of sigmoid is show in Figure 1.3.

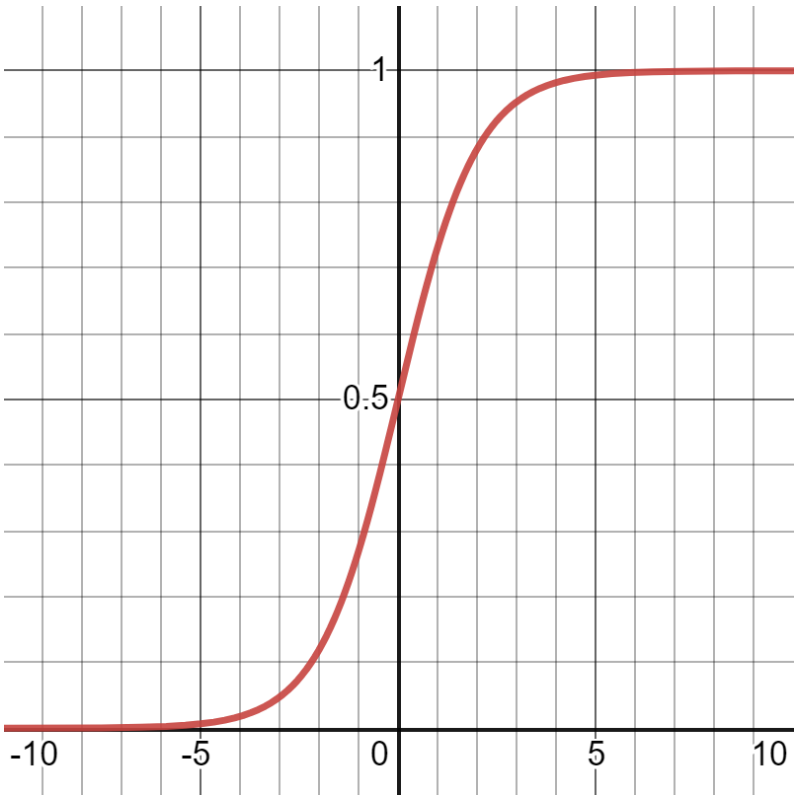

Fig. 1.3. Plot of sigmoid

Tanh is a non-linear function which is described by equation in Formula (1.6).

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{1.6}$$

where x – input to function;

e – exponent.

Interesting aspect of tanh that makes it more useful than some of its counterparts is that its range is $(-1, 1)$. As a result, the negative correlation becomes possible, which in turn frequently outperforms activation functions without negative values in range (Nielsen, 2015; Trask, 2019). The plot for the function is illustrated in Figure 1.4.

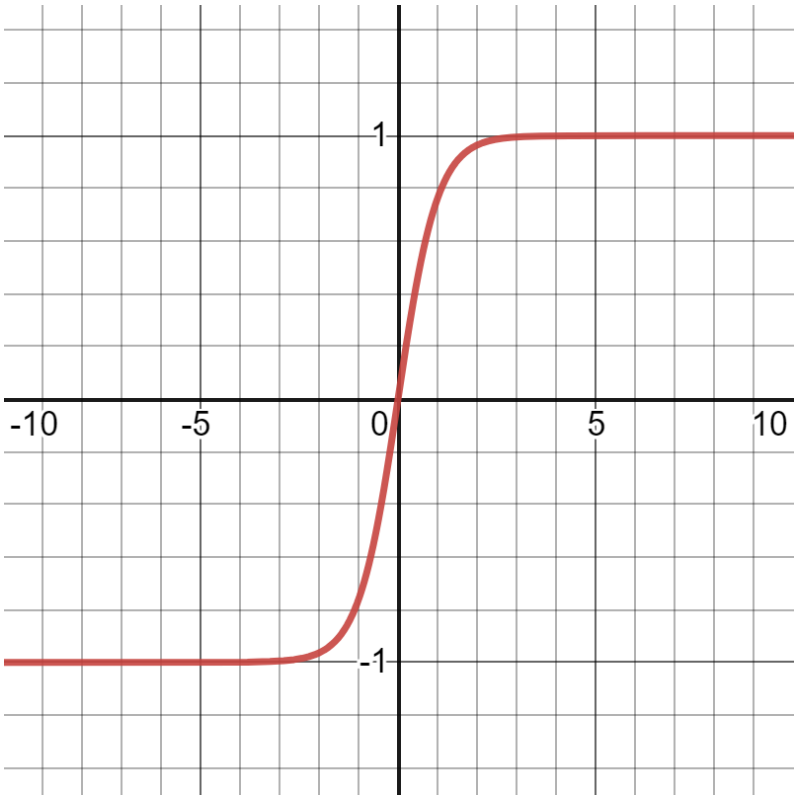

Fig 1.4. Plot of hyperbolic tangent

ReLu is another popular option for activation function. It is used as its range of values is $(0, \infty)$ and it is quick and simple in terms of computations (Aggarwal, 2018; Zhang *et al.*, 2021). Moreover, it has a property of executing layers with zero weights, therefore as stated in the book (Russell and Norvig, 2021) in the context of ReLu: "…they are believed to avoid the problem of vanishing gradients…" The computation of ReLu can be seen in Formula (1.7). The gradient will be discussed

later in this chapter.

$$f(x) = \begin{cases} x \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}, \tag{1.7}$$

where x – input to function.

The plot of the function is illustrated in Figure 1.5.

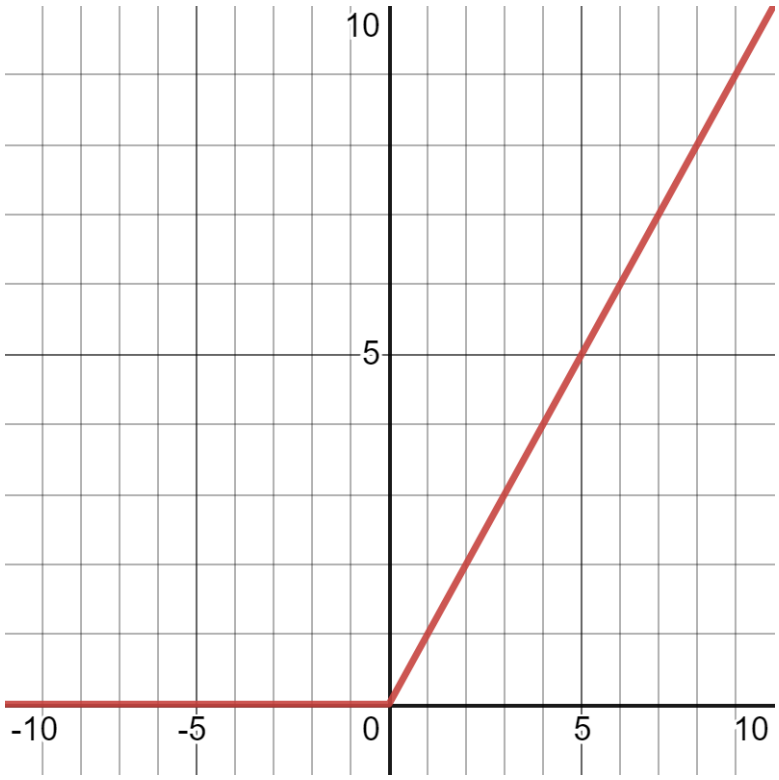

Fig. 1.5. Plot of ReLu

ReLu can encounter a problem in case it always has negative inputs, this will lead to ReLu "dying" and being unable to learn. For this reason LeakyReLu exists.

LeakyReLu is an activation function mostly similar to ReLu with a difference of a small scalar value a that is multiplied by input in case it is negative. This solves the problem of "dying", because now there is an output for negative values which can be used for learning (Zhang *et al.*, 2021). The LeakyReLu can be is defined by equation in Formula (1.8).

$$f(x) = \begin{cases} x \; if \; x > 0 \\ ax \; if \; x \leq 0 \end{cases}, \tag{1.8}$$

where x – input to function;

a – small scalar in range $(0, 1)$.

Plot for LeakyReLu is given in Figure 1.6.

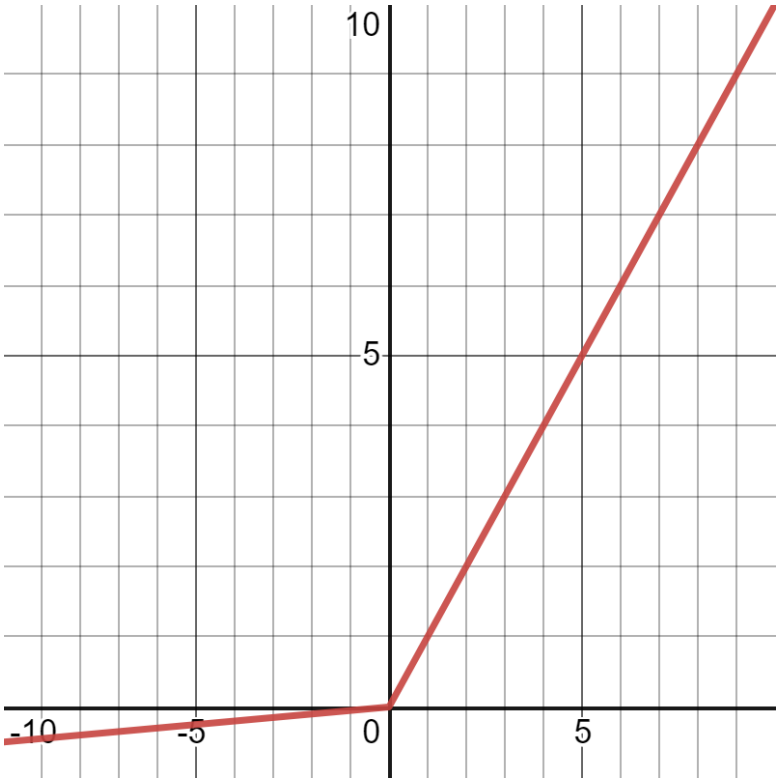

Fig. 1.6. Plot of LeakyReLu with $a = 0.05$

Another type of functions that can be encountered in neural networks are normalization functions. Normalization functions are used to avoid feeding the neural network with inputs that have different ranges of values. It helps network to learn and avoid situations in which one feature always affects the learning process of network more than other feature as a result of difference of value ranges. Usually, normalization is used to get the data into ranges $(0, 1)$ or $(-1, 1)$. For image-classification it is used to get the data from range $(0, 255)$ to (0, 1), this can be done as simple as dividing each value by 255. For RGB images the normalization is done across all input channels. Without normalization the pixels with higher intensities can skew the results, by diminishing other values. (Charniak, 2018; Vasilev *et al.*, 2019)

Two common ways to perform normalization are feature scaling and standard score. The difference is that feature scaling is performed on data that does not follow Gaussian distribution, while standard score on data that does. Feature scaling is defined by equation in Formula (1.9).

$$\boldsymbol{x = \frac{x - x_{min}}{x_{max} - x_{min}}}, \tag{1.9}$$

where x – single feature;

x_{min} – minimal value of the feature in the dataset;

x_{max} – maximal value of the feature in the dataset.

Standard score can be seen in Formula (1.10).

$$\boldsymbol{x = \frac{x - \mu}{\sigma}}, \tag{1.10}$$

where x – single feature;

$\boldsymbol{\mu}$ – mean of the features;

$\boldsymbol{\sigma}$ – standard deviation.

Another technique used for normalization is batch normalization. For context, batches are collections of input data of set size that dataset is divided into. Advantages of using batch normalization include adapting to the changes of the data overtime, which is done by internally preserving two variables. These two variables are exponential moving average of mean and variance. These variables are preserved separately for each batch (Chollet, 2018). The data is normalized before passing through activation functions (Buduma and Locascio, 2017).

1.2. Loss functions

Loss function is a fundamental element in neural network. It is used to calculate the distance between the expected and predicted output. The feedback of loss functions is used by optimizer to improve the performance of the algorithm. In simplest models, the weights are adjusted after processing every sample, these adjustments are done to minimize the loss function, which in turn means that expected and predicted outputs are getting closer. The loss function can be interpreted as measure of success, if it is low, it means that the task given to the neural network is performed accurately (Chollet, 2018). Therefore, minimization of loss function is one of the main tasks in machine learning.

One of the loss functions that are used for classification tasks is cross-entropy loss. It is computed by taking the negative logarithm of the prediction that corresponds to the expected value. This works, because negative sign will lead to lower result of loss function as the prediction for expected output gets higher.

Moreover, natural logarithm increases with higher inputs. This leads to the conclusion that for higher probabilities of correct result, the error will be lower (Charniak, 2018). The equation is given in Formula (1.11).

$$f(y, y') = -\sum_{i=0}^{n} y_i \cdot \log_e(y_i'), \tag{1.11}$$

where y_i – expected binary output for class i (0 or 1);

y_i' – predicted output for class i;

n – number of classes used in classification.

The sum in the equation is needed for multiclass classification problems, to sum errors over all class predictions.

Another type of cross-entropy loss is binary cross-entropy. It is utilized for two-class classifications (Vasilev *et al.*, 2019). The formula is simpler and does not need summation of errors. Binary cross-entropy can be seen in Formula (1.12).

$$f(y, y') = -y * \log_e(y') + (1 - y) * \log_e(1 - y'), \tag{1.12}$$

where y_i – expected binary output (0 or 1);

y_i' – predicted output.

1.3. Backpropagation algorithm

Backpropagation is "…the central algorithm in deep learning," according to book (Chollet, 2018). It is the algorithm used for computation of gradients. Gradients are used by optimizers to in neural network's learning process. The algorithm is called backpropagation as it propagates through the network from the output layer to the input layer while estimating the gradients for each layer. This is done by applying the calculus chain-rule (Goodfellow, Bengio and Courville, 2016; Vasilev *et al.*, 2019). These statements are substantiated by the quote from the book (Aggarwal, 2018) which states that "The backpropagation algorithm leverages the chain rule of differential calculus, which computes the error gradients in terms of summations of local-gradient products..." The steps of backpropagation algorithm are following (Zhang *et al.*, 2021):

1) calculate gradients of optimizer with respect to loss feedback and output layer,
2) calculate gradients with respect to parameters closest to output layer,

3) repeat calculation of gradients with respect to parameters for hidden layers until input layer is reached.

Simple example of chain rule for $F(x) = f(g(x))$ is presented in Formula (1.13).

$$\frac{dF}{dx} = \frac{d}{dx}[f(g(x))],$$
$$\frac{d}{dg(x)}[f(g(x))] \cdot \frac{d}{dx}[g(x)], \tag{1.13}$$

where $F(x)$ – function result;

x – input variable;

$f(g(x))$ – function with input $g(x)$;

$g(x)$ – function with input variable x.

1.4. Optimization algorithms

Optimizer is another crucial element in neural networks. Its purpose is to update weights based on the feedback received from loss function to maximize the efficiency of the model. Currently the most efficient optimization algorithms are gradient-based algorithms such as stochastic gradient descent (SGD) and its extensions. Algorithm iteratively looks for minimal value in gradients received from backpropagation algorithm to minimize the error. SGD descent updates the network after processing every training variable. However, this may lead to noisy data (Buduma and Locascio, 2017; Chollet, 2018).

To avoid noise in data the mini-batch gradient descent is used. Idea behind mini-batches is the division of dataset into batches of set size. Updates in mini-batch gradient descent are performed after, processing the whole mini-batch and accumulating its error. This reduces the noise in data, as the mean of gradients is used which balances the outlier training samples (Vasilev *et al.*, 2019). The equation to calculate the mean of gradients for minibatch is represented by Formula (1.14) (Nielsen, 2015).

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}, \tag{1.14}$$

where ∇C – estimated gradient;

m – number of samples in mini-batch;

∇C_{X_j} – gradient for specific input.

Example of SGD in 1D plane is illustrated in Figure 1.7.

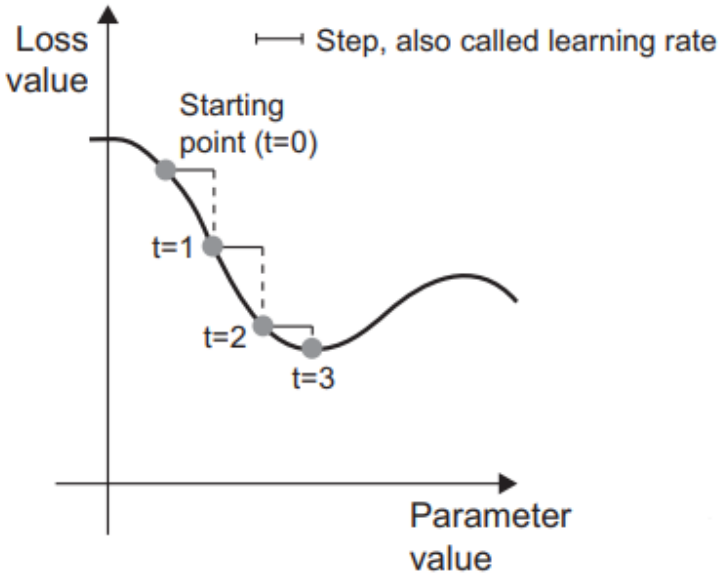

Fig. 1.7. SGD with one learnable parameter (adopted from (Chollet, 2018))

Example of gradient descent in a 3D plane is illustrated in Figure 1.8.

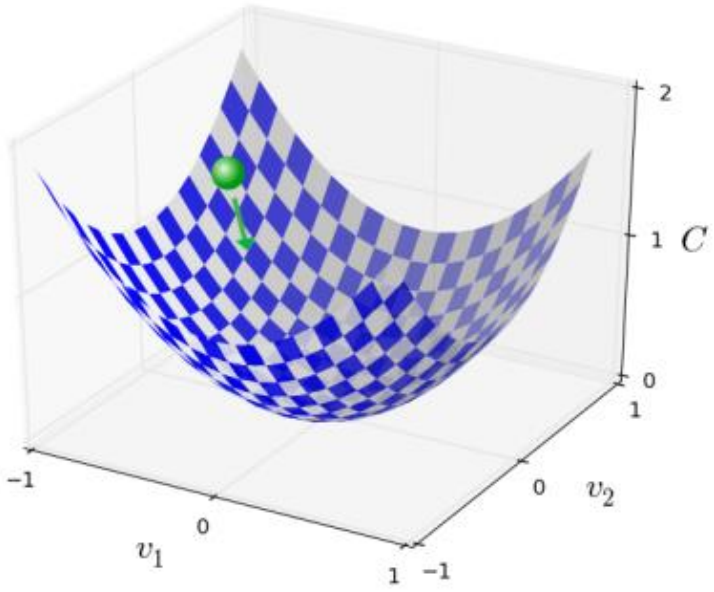

Fig. 1.8. Gradient descent in a 3D plane (adopted from (Nielsen, 2015))

1.5. Metrics

To measure the performance of the algorithm different metrics are used. In case of classification – accuracy, precision, recall and F1-score are metrics that may be used to represent how efficient is the model. To count these metrics, first the confusion matrix is calculated. Confusion matrix serves a role of summarizing the results of classification. From confusion matrix, four metrics can be derived. True positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). In confusion matrix for binary classifications, there would be four cells – one for each of these metrics. The reason for this is that with binary classification, one class can be treated as positive and other as negative (Sammut and Webb, 2017). Example of confusion matrix for binary classification can is shown in Figure 1.9.

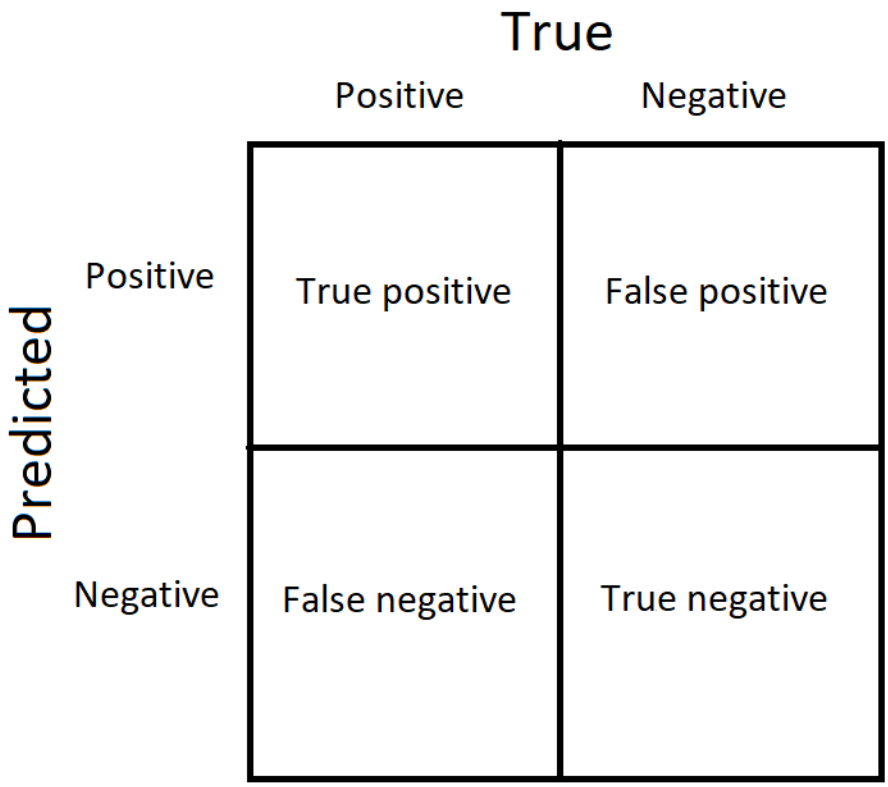

Fig. 1.9. Confusion matrix for binary classification

True positive means that both predicted and true class are positive. False positive means that predicted class is positive while true class is negative. False negative means that true class is positive, while predicted is negative. True negative means that both predicted and true class are negative (Sammut and Webb, 2017). In multiclass classification these metrics are counted with respect to each class.

In multiclass classification confusion matrix represents which class was predicted for which true class. This data can be used to count TP, TN, FP, FN applying similar concepts to binary classification. To do that only the class with respect to which the metrics are currently counted has to be treated as positive and all others as negative. Example of confusion matrix for multiclass classification can be seen in Figure 1.10. Numbers in cells of matrix are in percentages. For example, cell in first row and first column means that when “T-shirt/top” was True class, it was predicted in 40% of cases.

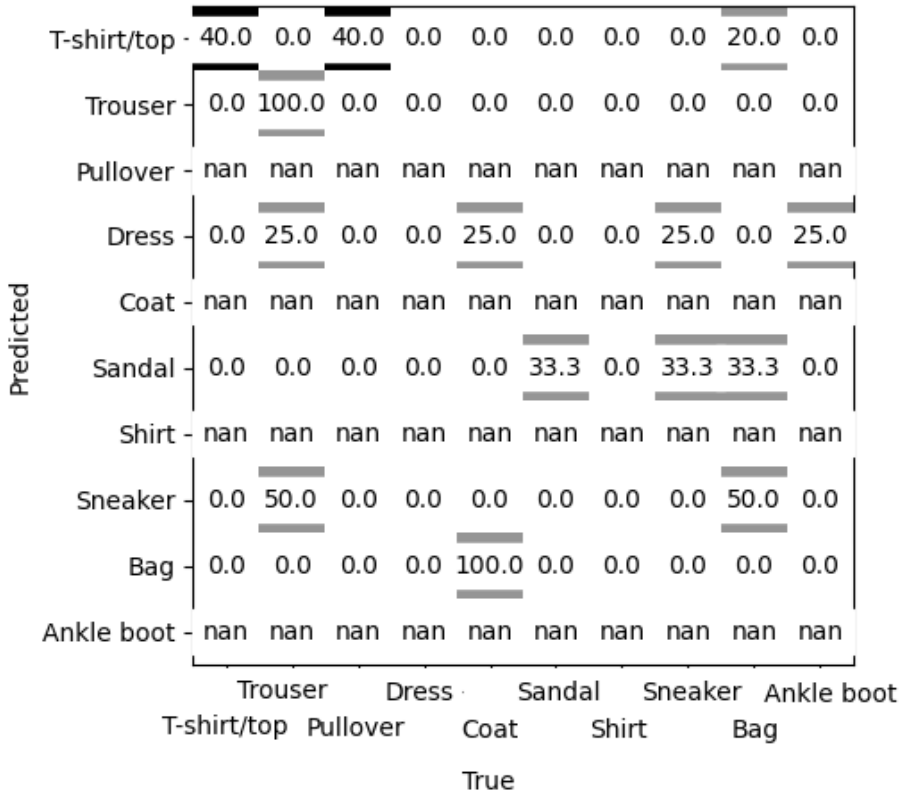

Fig. 1.10. Confusion matrix for multiclass classification

Metrics derived from confusion matrix can be used to count other metrics, such as, accuracy, recall and precision.

Accuracy is a general metric which represents percentage of instances of the dataset that are classified correctly. This metric represents whole dataset and is not related to specific class (Sokolova, Japkowicz and Szpakowicz, 2006). Accuracy is calculated by taking number of correct classifications and dividing it by total number of objects. For binary classification calculation is represented by Formula (1.15).

$$acc = \frac{tp + tn}{tp + fp + fn + tn}, \tag{1.15}$$

where acc – accuracy;

tp – number of true positives;

tn – number of true negatives;

fp – number of false positives;

fn – number of false negatives.

For multiclass classification, to calculate accuracy sum of TPs with respect to each class are divided by sum of all cells of confusion matrix.

Precision is a metric specific to each class. Precision represents the fraction of correct predictions with respect to the class regarding all times this class was predicted (Sokolova, Japkowicz and Szpakowicz, 2006). Precision is calculated by Formula (1.16).

$$pre = \frac{tp}{tp + fp}, \tag{1.16}$$

Where pre – precision;

tp – number of true positives;

fp – number of false positives.

Recall is another class-specific metric. It represents the fraction of correct predictions for the class out of all predictions that were made while the class was positive (Sokolova, Japkowicz and Szpakowicz, 2006). Recall is calculated by Formula (1.17).

$$rec = \frac{tp}{tp + fn}, \tag{1.17}$$

where rec – recall;

tp – number of true positives;

fn – number of false negatives.

F-measure is a metric that as described by (Liu, 2018) as "…a compromise between recall and precision." It is defined as a harmonic mean of precision and recall. Calculation of F-measure is done by Formula (1.18).

$$F = 2 \cdot \frac{pre \cdot rec}{pre + rec} = \frac{2tp}{2tp + fp + fn}, \quad (1.18)$$

where F – F-measure;

pre – precision;

rec – recall;

tp – number of true positives;

fp – number of false positives;

fn – number of false negatives.

2. DEEP LEARNING CLASSIFICATION

Deep learning is often used for classification tasks. The main goal of classification is maximization of accuracy of predictions of the model. Given a set of inputs the model should be able to correctly determine to which of the predefined labels those inputs correspond (Charniak, 2018). The simplest form of classification is binary classification which revolves around predicting inputs for two classes. Example of binary classification is the image classification task in which the model must classify the image into two categories cat or dog. More complex form of classification is multiclass classification. Multiclass classification revolves around categorizing inputs into 3 or more classes. Example of multiclass classification is handwritten digits classification task, which has 10 classes which correspond to numbers 0-9 (Zhang *et al.*, 2021).

2.1. Inputs

Inputs in deep learning classification consist of data and its labels, for learning process, or just data needed for classification in case the model is already trained. Inputs for different classification tasks differ, for example, for the face-recognition task the inputs would be labeled faces of people, whereas for speech-recognition audio files of human speech, with attached human created transcripts (Chollet, 2018).

As mentioned previously, inputs may come in different forms and therefore they are preprocessed and stored differently.

Easiest setup of inputs for classification is vectors and related scalar labels. Vectors consist of scalar values which are inserted into the model as a set of features for which the weights must be derived (Chollet, 2018; Trask, 2019). For that reason, it can be stated that scalar inputs are the simplest form of inputs that model receives.

Inputs may also be categorical, which means that the input feature is not numeric but belongs to some category. This may pose problems as often there is no order in categories, therefore, the categories cannot be represented as numbers $[1 \dots m]$ where m is the number of categories. This ordinal notation may be used in cases where the categories have a relationship, otherwise it would skew the results, as the higher number would be treated differently. This leads to one-hot encoding, which means encoding categories as a binary vector. The correct category would be set to 1 and all other categories to zero (Vasilev *et al.*, 2019; Zhang *et al.*, 2021). Example of

one-hot encoding is encoding cat, dog and chicken as a vector in which, first element corresponds to cat, second element corresponds to dog and third element corresponds to chicken, this means that to encode dog vector $[0, 1, 0]$ is created. For cat and chicken vectors $[1, 0, 0]$ and $[0, 0, 1]$ respectively.

A visualization of scalar notation of inputs and respective vector notation with same number of inputs can be seen in Figures (2.1; 2.2).

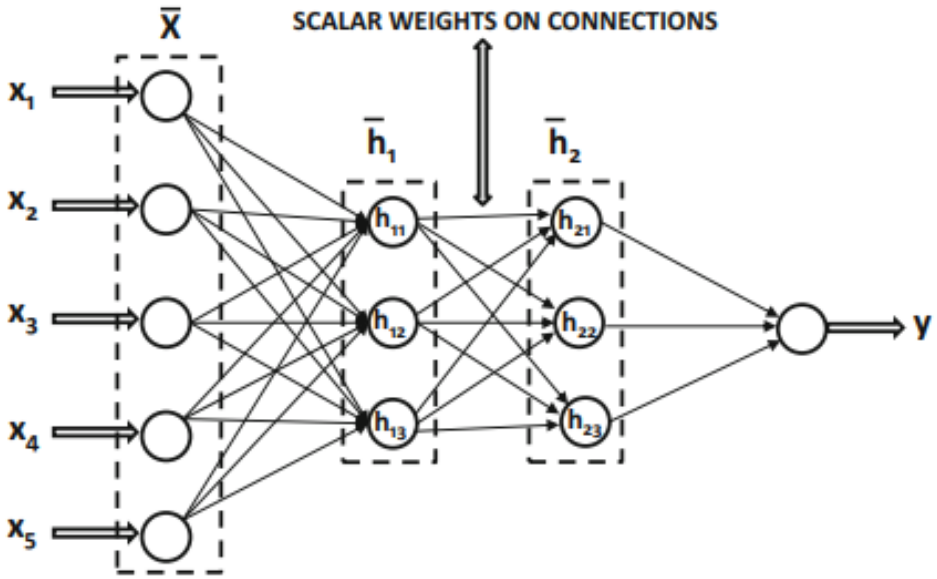

Fig. 2.1. Scalar notation of inputs in a model with two hidden layers (adopted from (Aggarwal, 2018))

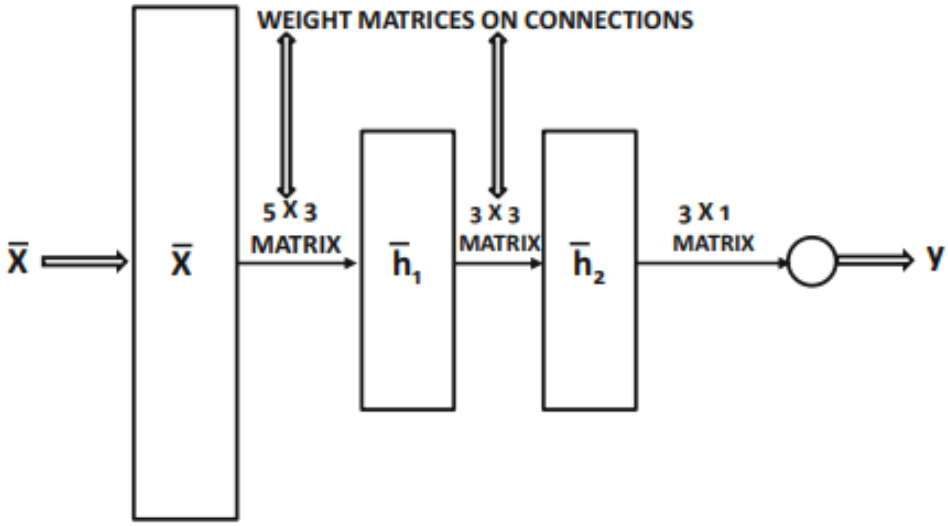

Fig. 2.2. Vector notation of inputs in a model with two hidden layers (adopted from (Aggarwal, 2018))

Another type of inputs are image inputs. Image inputs may differ in dimensionality, usually it is a 3D input consisting of width, height, and number of color channels, however, this depends on the number of color channels that image has. If the image is greyscale, then it can be stored as a 2D object as it has single color channel (Chollet, 2018). Image is represented by a set of pixels which have values that correspond to specific color and its intensity in a range $[0, 255]$. For greyscale image it can be treated as intensity of black colour where 0 is white and 255 is black (Vasilev *et al.*, 2019). For coloured images there are several input channels, for example, RGB images there are 3 input channels, each channel corresponds to either red, green, or blue. The number of input channels affects the number of input features. Image with resolution of 100×100 with 3 input channels would consist of 30000 input features. It is important to note that in image classification task each pixel corresponds to a feature (Vasilev *et al.*, 2019; Zhang *et al.*, 2021). Figures (2.3; 2.4) show an example of grayscale image from MNIST handwritten digits dataset and its representation as an array of numerical values. Images in dataset have 28×28, which means that each image consists of 784 pixels.

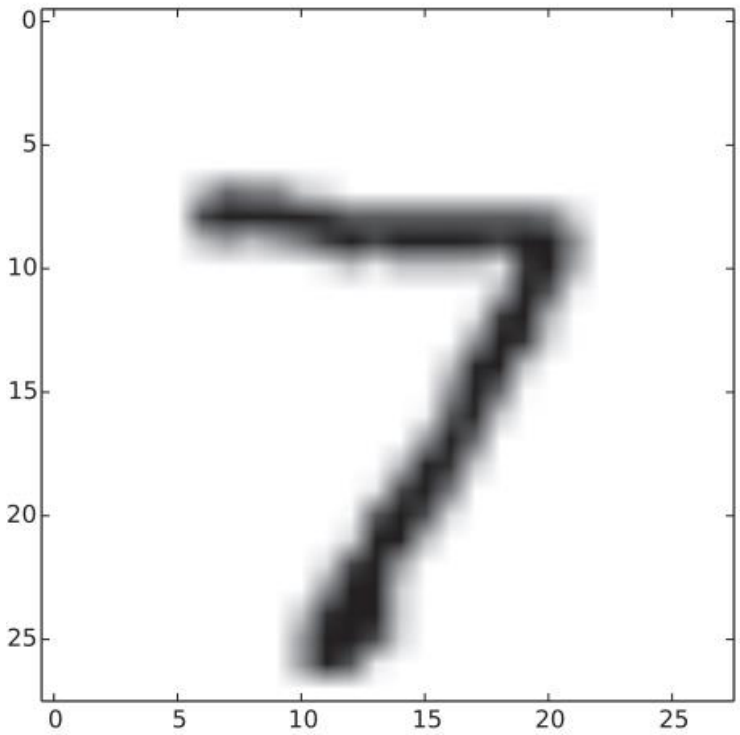

Fig. 2.3. Greyscale image of 7 from MNIST handwritten digits dataset (adopted from (Charniak, 2018))

	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	185	159	151	60	36	0	0	0	0	0	0	0	0	0
8	254	254	254	254	241	198	198	198	198	198	198	198	198	170
9	114	72	114	163	227	254	225	254	254	254	250	229	254	254
10	0	0	0	0	17	66	14	67	67	67	59	21	236	254
11	0	0	0	0	0	0	0	0	0	0	0	83	253	209
12	0	0	0	0	0	0	0	0	0	0	22	233	255	83
13	0	0	0	0	0	0	0	0	0	0	129	254	238	44
14	0	0	0	0	0	0	0	0	0	59	249	254	62	0
15	0	0	0	0	0	0	0	0	0	133	254	187	5	0
16	0	0	0	0	0	0	0	0	9	205	248	58	0	0
17	0	0	0	0	0	0	0	0	126	254	182	0	0	0
18	0	0	0	0	0	0	0	75	251	240	57	0	0	0
19	0	0	0	0	0	0	19	221	254	166	0	0	0	0
20	0	0	0	0	0	3	203	254	219	35	0	0	0	0
21	0	0	0	0	0	38	254	254	77	0	0	0	0	0
22	0	0	0	0	31	224	254	115	1	0	0	0	0	0
23	0	0	0	0	133	254	254	52	0	0	0	0	0	0
24	0	0	0	61	242	254	254	52	0	0	0	0	0	0
25	0	0	0	121	254	254	219	40	0	0	0	0	0	0
26	0	0	0	121	254	207	18	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 2.4. Image as multidimensional array of numerical values (adopted from (Charniak, 2018))

2.2. Probability distribution

In classification tasks the output is an estimation that input resembles specific class. This is done by converting data using probability distribution layers. Values that go into probability distribution function are typically called logits. Logits are transformed by function into a set of non-negative numerical values which sum up to one (Charniak, 2018). In this set of numerical values, each value represents probability that input is of respective class. It is done this way as it is highly unlikely that some class will be predicted with 100% confidence (Buduma and Locascio, 2017).

One of the functions used to acquire probability distribution is softmax. Each output in the softmax depends on all other outputs of the layer. This is the result of the probability distribution having to sum up to 1 (Trask, 2019; Vasilev *et al.*, 2019). The softmax is calculated by Formula (2.1).

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \tag{2.1}$$

where $f(x)_i$ – i-th output of softmax layer;

e – Euler's number;

x_i – i-th logit;

$\sum_{j=1}^{N} e^{x_j}$ – sum of all logits.

It is guaranteed that outputs of softmax will be positive as e^x is always positive. Softmax name comes from its property to highlight one value as the highest leading to strong predictions. As it is described in the book (Charniak, 2018), "…"softmax" gets its name from the fact that it is a "soft" version of the "max" function. The output of the max function is completely determined by the maximum input value. Softmax's output is mostly but not completely determined by the maximum."

As softmax has a lot of variables it is difficult to plot it. However, if all variables except for one are treated as constants, it can be plotted. Therefore, in situation where logit x_1 varies and logits x_2, x_3 are equal to 3 and -4 respectively, it can be seen in Figure 2.5., how the predicted probability varies depending on value of x_1. For $x_1 = 3$ probability would be close to 0.5 as $x_2 = 3$ and $x_3 = -4$. For values higher than three x_1 probability starts to rise, while probability for other values lowers.

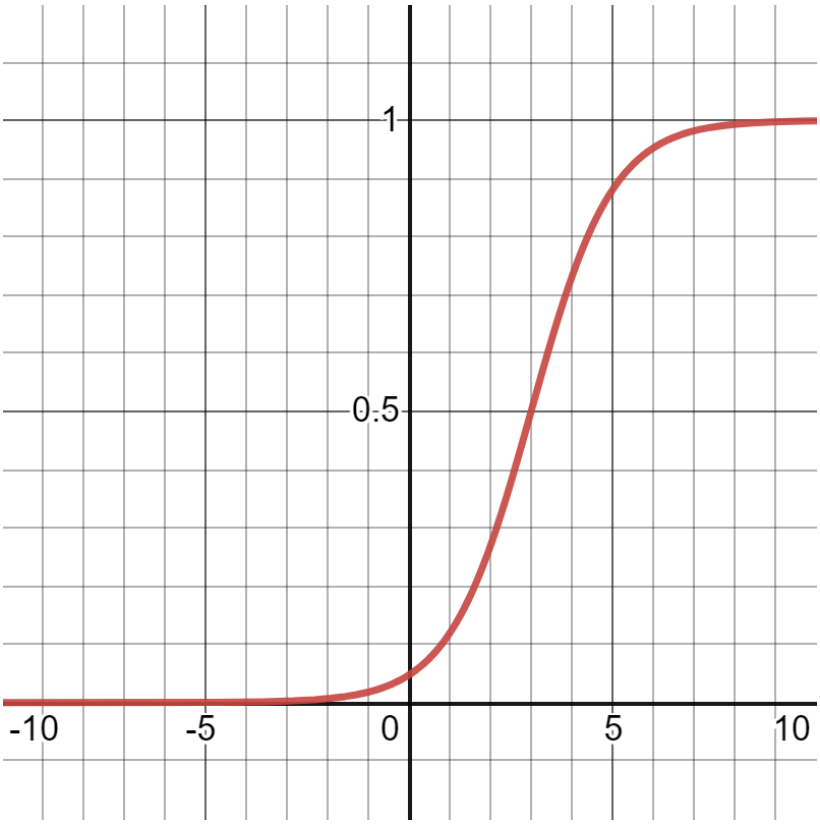

Figure 2.5. Plot of softmax with respect to x_1

2.3. Image classification layers

To understand architecture of image-classification neural network, the layers used must be described.

Fully connected layers or dense layers are those that connect each of its units to each of the units in the next layer (Nielsen, 2015). In Keras dense layer is implemented by performing a transformation of inputs using linear function and then activating them with activation function of choice (Chollet, 2015).

Convolutional layers are widely used in image-classification tasks. They are used to apply filters to input images to create so called feature maps. The filters are smaller than input, which allows to capture specific features. A filter consists of learnable weights and is applied to different parts of an image. The weights of the filter are shared for the whole image, to highlight the specific feature in all parts of the image. Different filters are applied to images to detect various features and to learn to discern them. In most cases, different filters are applied to same image, to find various features. This means that more feature maps are created (Buduma and Locascio, 2017).

The filters are moved using stride hyperparameter, stride determines how much the filter is moved from its previous position and therefore dictates how many different parts of the image will be captured. For RGB images, each channel has a separate filter. The output is calculated by multiplying filter weights by respective parts of an image and summing them up. If the filters are applied across different color channels, the sum of the output of each channel is taken. As a result of applying filters the output size is smaller than input. In such cases to have more control of the output size, padding can be applied. The padding consists of values equal to 0 which are stationed around the edges of an image (Buduma and Locascio, 2017; Vasilev *et al.*, 2019). Figure 2.6. is an example of application of filters across a 5×5 input with 3 channels and padding 1. Two 3×3 filters with 3 channels are used. The acquired output are 2 feature maps as the stride is set to 2.

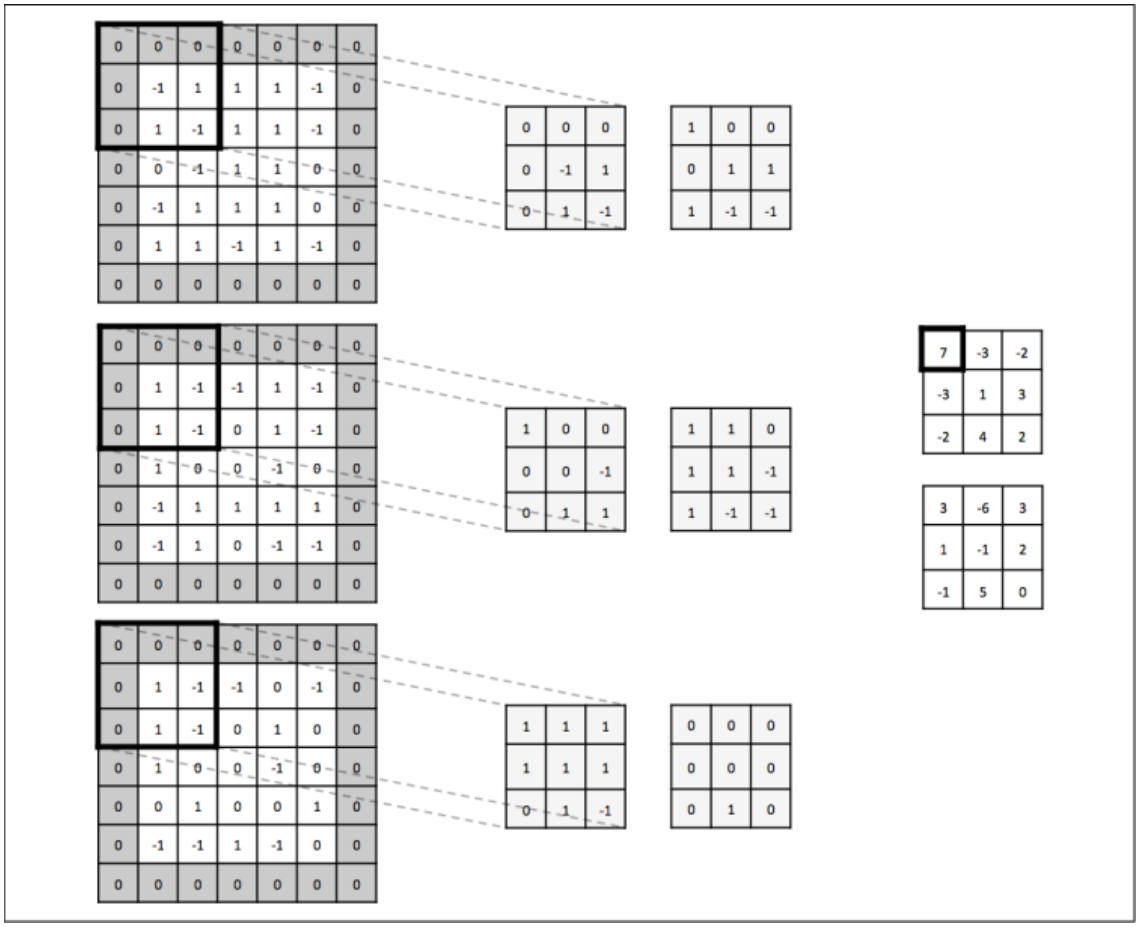

Fig 2.6. Example of application of convolutional layer (adopted from (Buduma and Locascio, 2017))

Another type of layers are pooling layers, in this case max-pooling (MP) specifically. They are used to reduce the size of an output from convolutional layer. In max pooling this is done by dividing the feature map into sections of similar size, finding the maximal value among the values, and propagating it into the condensed version of the image. Max-pooling layer has its own stride, width, and height to determine the number of sections that the image will be divided into and how to traverse it. Pooling layers do not have learnable parameters (Nielsen, 2015; Vasilev *et al.*, 2019). MP layers have an interesting property which is described in the book (Buduma and Locascio, 2017) as "One interesting property of max pooling is that it is locally invariant. This means that even if the inputs shift around a little bit, the output of the max pooling layer stays constant." Which means that that even if initially features are encountered in slightly different parts of the image, they will anyway be mapped to same pixel after MP layer. Figure 2.7 illustrates an example of 2×2 max pooling with stride 2 applied to a 4×4 input. The output is 2×2, thus the input was

reduced by 4 times.

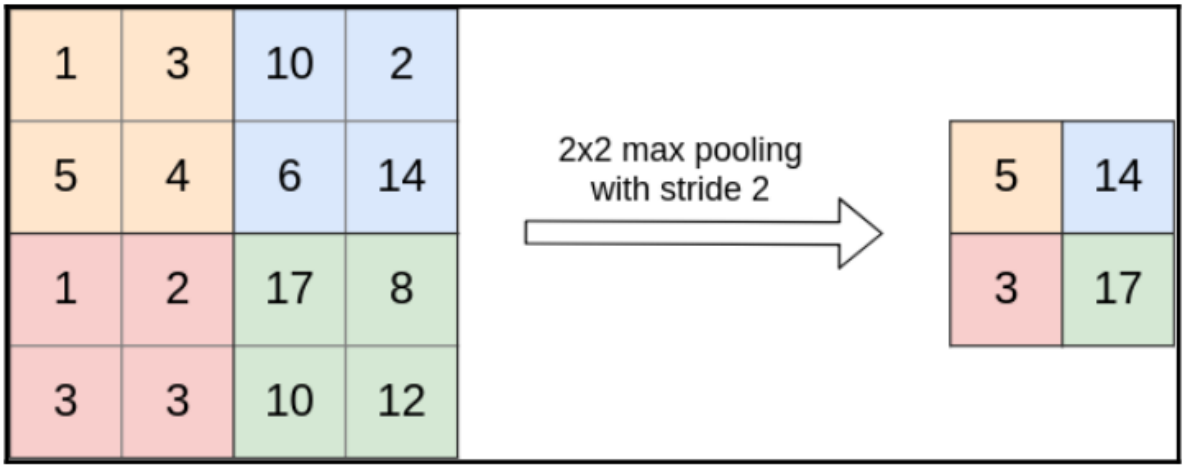

Fig 2.7. Example of application of max pooling layer (adopted from (Vasilev *et al.*, 2019))

2.4. Example architecture

As an example of architecture for classification task the architecture of AlexNet will be used. AlexNet is a winner of 2012 ImageNet Large Scale Visual Recognition Challenge. Initially AlexNet had to be processed using two GPUs, however for simplicity of understanding the architecture will be shown as a singular process. AlexNet features convolutional layers, max pooling layers, fully connected layers and softmax. ReLu activation functions are used in between convolutions (Aggarwal, 2018).

Layers in AlexNet are organized as shown in Table 2.1.

Table 2.1

AlexNet Architecture (adapted from (Aggarwal, 2018))

Input size	Operation
$224 \times 224 \times 3$	Convolution 11×11 at stride 4 with 96 filters
$55 \times 55 \times 96$	Convolution 5×5 at stride 2 with 256 filters and max pooling 3×3 at stride 2
$27 \times 27 \times 256$	Convolution 3×3 at stride 2 with 384 filters and max pooling 3×3 at stride 2
$13 \times 13 \times 384$	Convolution 3×3 with 384 filters at stride 2

Input size	Operation
13 × 13 × 384	Convolution 3 × 3 with 256 filters at stride 2
13 × 13 × 256	Max pooling 3 × 3 at stride 2
4096	Fully connected
4096	Fully connected
1000	Softmax

AlexNet has eight weighted layers, five of them are convolutional and three are fully connected. The activation function for output layer is softmax, which creates a probability distribution with 1000 outputs (Krizhevsky, Sutskever and Hinton, 2017). Figure 2.8 provides visualization of AlexNet architecture.

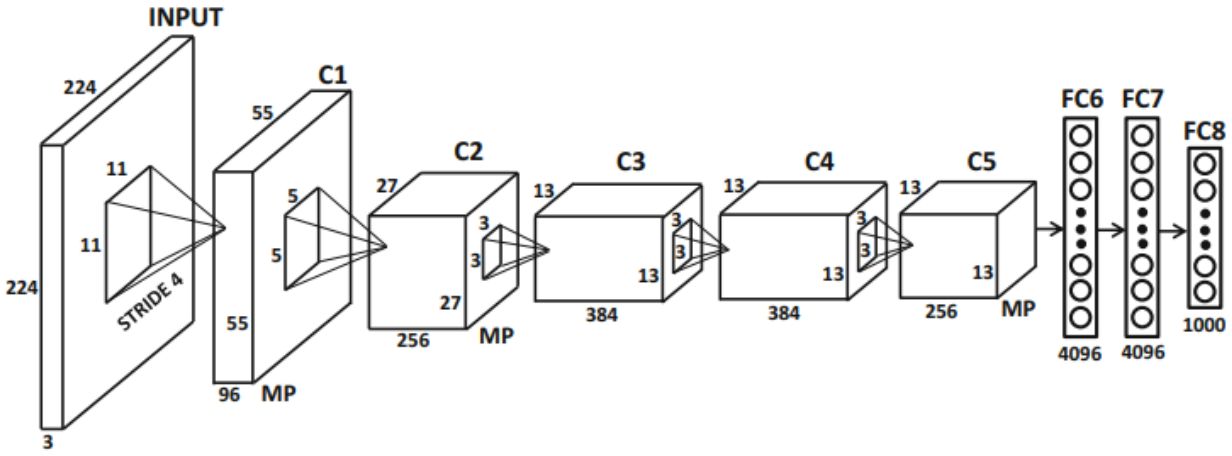

Fig 2.8. AlexNet architecture (adopted from (Aggarwal, 2018))

AlexNet was chosen as an example of architecture, as it features many design conventions that were not popular before it. As examples, data augmentation, ReLu usage, usage of GPUs for training are given (Aggarwal, 2018). Thus, it can be stated that AlexNet is a significant example of convolutional network architecture.

3. METHODOLOGY

To compare the performance and possibilities of application for different loss functions for image-classification it is needed to perform experiments. Experiments must be approached in a justified manner to acquire reliable data. For the experiments, the Python programming language is used, as it is supported by a number of tools that are useful for deep learning.

Tools that are used in experiments and their functionality:

1) PyTorch is a machine learning framework that eases the process of development of neural network;
2) NumPy is a library used for numerical computing;
3) TensorBoard is a machine learning visualization tool with rich functionality, which allows to store and compare visualizations of different runs as well as their parameters.

The experiments are performed using two NVIDIA Quadro 4000 GPUs. The coding is done in PyCharm integrated development environment.

As a model pretrained ResNet50 is used, models with higher number of layers were not chosen because of time constraints.

Adam optimizer is used, it is more flexible and maintains learning rates that can adapt for each feature in the network. Adam has shown its efficiency in comparison to other optimizers (Kingma and Ba, 2017).

Number of transformations is applied to images to diversify the dataset while training, thus increasing the accuracy of predictions. Transformations are applied in a following order:

1) image is horizontally flipped with probability 50%;
2) image is vertically flipped with probability 50%;
3) image is rotated in range $[-90, 90]$ degrees and scaled in range $[0.8, 1.2]$ with probability 90%;
4) image brightness, contrast, saturation and hue are increased by 0.2 with probability 40%.

Following hyperparameters are encountered in the experimental program:

1) learning rate,
2) batch size,
3) gamma (for focal loss),

4) margin (for soft-margin softmax).

To determine suitable hyperparameters for experimentation, two approaches were used.

Batch size and learning rate were chosen through experiments with different configurations on parts of datasets. Batch size 32 was chosen as it is high enough to train the network effectively and low enough to fit into GPU memory. For learning rate 1e-3, 1e-4 and 1e-5 were tested. 1e-3 proved to be too high, and loss started to diverge, while 1e-5 was too low, so the results were unsatisfactory. Therefore, 1e-4 was chosen as it proved to be stable, and the results proved to be satisfactory.

Gamma (for focal loss) and margin (for soft-margin softmax) values are taken as suggested in respective papers. For gamma 0.5 and 2 are chosen. As a margin 0.3 is taken.

There are 45 experimental configurations, 3 datasets with 4 loss functions (focal loss will be with two different gammas), with 3 probability distribution functions.

Experimental results are compared with respect to their performance with different datasets and probability distribution functions. Each dataset features 3 experiments with different probability distribution functions for each of 5 possible variations of loss functions. These experiments are compared in sets of 3 to determine which loss functions works better with which probability distribution function, moreover the configuration with best accuracy for each dataset are found as well. The confusion matrixes are analysed to see if there are complexities with classification for specific classes. Main metrics that are used for comparison are best test accuracy and the number of epochs in which this accuracy was achieved.

Moreover, the loss functions are compared using the placement system, as there are 5 variations of loss functions, depending on the speed of convergence and the best accuracy they will be placed into one of five places, and provided according to number of points, with 1^{st} place getting five points and 5^{th} place getting 1 point. The placement will be decided for each dataset. The points are summed up and provide additional metric which can be used to compare the loss functions.

To prove the statistical significand or insignificance of the results, the z-score and t-score are used.

Code files used for experiments are accessible through the GitHub link: Code.

3.1. Loss functions for classification

There are three loss functions that are compared in this research.

The experiments with different loss functions are done with sets of different hyper-parameters. This allows to find correlations between changes in specific hyper-parameters and efficiency of loss functions. Moreover, as the datasets consist of different it is possible that some loss function performance is affected more by initial resolution of images or number of input channels. This is a possibility, as different loss functions produce different output for similar inputs. Therefore, one loss function may treat features derived from one dataset better than other loss function and vice versa.

To calculate the loss, instead of one-hot encoding indexes of true classes are stored in a separate array to decrease memory and computational complexity.

Loss functions that are compared are following:

1) categorical cross-entropy loss,
2) focal loss,
3) Kullback-Leibler divergence (KL divergence),
4) Jensen-Shannon divergence (JS divergence).

3.1.1. Focal loss

Focal loss is a loss function designed to address the problem of class imbalance. In comparison, cross-entropy loss is overwhelmed by class imbalance. Focal loss is based on cross-entropy loss, but it adds a modulating factor shown in Formula (3.1) (Lin *et al.*, 2018).

$$(1 - y_i')^{\gamma}, \tag{3.1}$$

where y_i' – predicted output for class i;

γ – focusing parameter.

Therefore, adding the modulating factor to cross-entropy loss, the calculation for focal loss is defined as Formula (3.2). When γ equals 0, the focal loss is equal to cross-entropy loss.

$$f(y, y') = -\sum_{i=0}^{n} y_i \cdot (1 - y_i')^{\gamma} \cdot \log_e(y_i'), \tag{3.2}$$

where y_i – expected binary output for class i (0 or 1);

y_i' – predicted output for class i;

γ – focusing parameter;

n – number of classes used in classification.

3.1.2. KL divergence

To measure the difference between two probability distributions, KL divergence can be used. KL divergence calculates how the expected output from the model diverges from true output. KL diverges largely resembles cross entropy (Murphy, 2012). KL divergence is calculated as seen in Formula (3.3). KL divergence can be used as alternative of loss function in classification tasks as their output is probability distribution.

$$f(y, y') = -\sum_{i=0}^{n} y_i \cdot \log_e \left(\frac{y_i}{y_i'}\right), \tag{3.3}$$

where y_i – expected binary output for class i (0 or 1);

y_i' – predicted output for class i;

n – number of classes used in classification.

3.1.3. JS divergence

One of the losses in this work is KL divergence. The symmetrized and smoothed version of it is called JS divergence. JS divergence is a squared metric, therefore the square root of JS divergence has to be taken to get the metric itself. (Fuglede and Topsoe, 2004). Formula (3.4) represents Jensen-Shannon divergence.

$$f(y, y') = \frac{1}{2} KL(y||m) + \frac{1}{2} KL(y'||m) \tag{3.4}$$

where y_i – expected binary output for class i (0 or 1);

y_i' – predicted output for class i;

m – $\frac{1}{2}(y + y')$;

$KL(y||m)$ – KL divergence with y as true distribution and m as approximation;

$KL(y'||m)$ – KL divergence with y' as true distribution and m as approximation.

3.2. Datasets

In this research two different datasets are used. The datasets differ in number

of classes, colour channels and resolution of images.

3.2.1. Kuzushiji MNIST

Kuzushiji MNIST dataset focuses on cursive Japanese script. The dataset was created by National Institute of Japanese Literature. The dataset consists of ten characters, representing 10 rows of Hiragana (part of Japanese writing system). Dataset, similar to MNIST contains 70000 images, with 7000 images per character. Each character has 6000 training and 1000 test examples. Images are greyscale, their resolution is 28×28. Images are labeled with names from Hiragana (Clanuwat *et al.*, 2018).

Dataset features listed classess, classes will be given as their transcription from Japanese.

1. O.
2. Ki.
3. Su.
4. Tsu.
5. Na.
6. Ha.
7. Ma.
8. Ya.
9. Re.
10. Wo.

Featured classes and their samples are illustrated in Figure 3.1.

Hiragana	Unicode	Samples	Sample Images
お (o)	U+304A	7000	
き (ki)	U+304D	7000	
す (su)	U+3059	7000	
つ (tsu)	U+3064	7000	
な (na)	U+306A	7000	

Hiragana	Unicode	Samples	Sample Images
は (ha)	U+306F	7000	
ま (ma)	U+307E	7000	
や (ya)	U+3084	7000	
れ (re)	U+308C	7000	
を (wo)	U+3092	7000	

Fig. 3.1. Samples and names of characters in the dataset (adopted from (Clanuwat *et al.*, 2018))

3.2.2. Fashion MNIST

Fashion Modified National Institute of Standards and Technology (Fashion MNIST) dataset consists of 70000 unique images of different articles of clothing. The dataset is developed using Zalando clothing catalogue. Dataset is divided into two subsets, the training subset which includes 60000 examples and test subset which includes 10000 examples. The image resolution is 28×28. Images are grayscale. Each image is labeled as one of the ten possible classes (Xiao, Rasul and Vollgraf, 2017). The dataset is balanced and has 7000 images of each class.

Dataset features listed classes.

1. FT-Shirt/Top.
2. Trouser.
3. Pullover.
4. Dress
5. Coat.
6. Sandals.
7. Shirt.
8. Sneaker.
9. Bag.
10. Ankle Boot.

An example of images from MNIST dataset is illustrated in Figure 3.2. Each two rows correspond to one class in the order that is listed above.

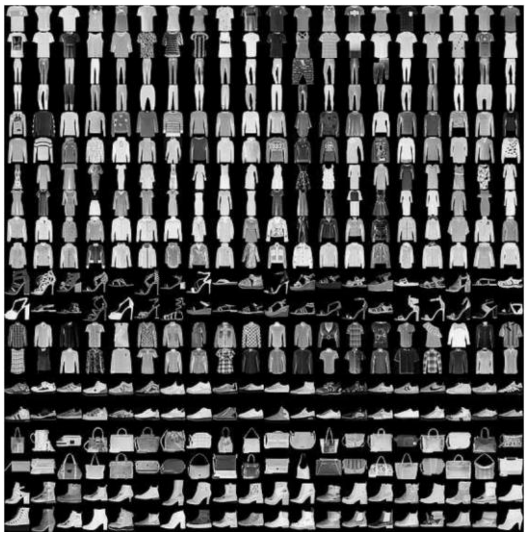

Fig. 3.2. An example of images from Fashion MNIST dataset (adopted from (Xiao, Rasul and Vollgraf, 2017))

3.2.3. CIFAR-10

Canadian Institute For Advanced Research 10 (CIFAR-10) dataset consists of 60000 images of either transport or animals. The dataset consists of labeled images from 80 million tiny images dataset. The dataset is divided into two subsets, the training subset which includes 50000 examples and test subset which includes 10000 examples. The image resolution is 32×32. Images are colored (Krizhevsky, 2009). There are 10 classes in this dataset, 4 classes for transport and 6 classes for animals. There are 6000 images of each class, therefore dataset is balanced.

Dataset features listed classes.

1. Airplane.
2. Automobile.
3. Bird.
4. Cat.
5. Deer.
6. Dog.
7. Frog.
8. Horse.
9. Ship.
10. Truck.

An example of images from the dataset (Fig. 3.3). Each row corresponds to one class in the order that is listed above.

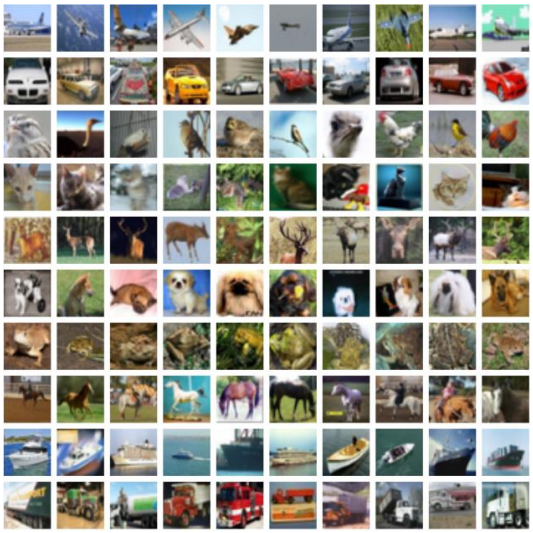

Fig. 3.3. An example of images from CIFAR-10 dataset (adopted from (Krizhevsky, Nair and Hinton, 2009))

3.3. Architecture

For purposes of this research Residual Network 50 (ResNet50) pretrained model is used. The ResNet models were developed in Microsoft Research by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. ResNet models vary in depth with shallowest model being 18 layers deep and deepest 152 layers deep. ResNet models use skip connections, so that inputs to one layer can be copied and added to output of another layer (He *et al.*, 2015). This in turn establishes iterative view of feature engineering in contrast to the usual hierarchical view (Aggarwal, 2018). Example of such skip is illustrated in Figure 3.4.

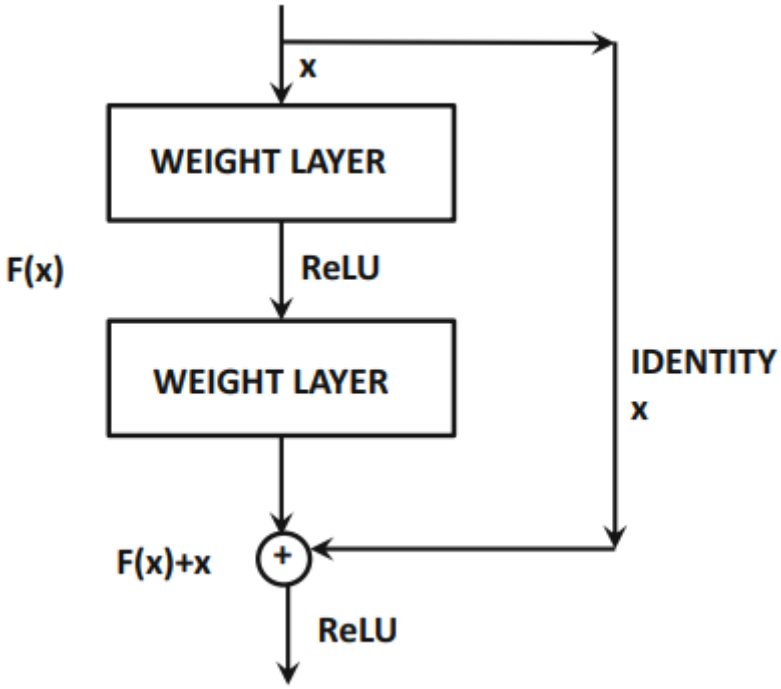

Fig. 3.4. Example of skip connection (adopted from (Aggarwal, 2018))

These skip connections are needed to establish an effective gradient calculation, as backpropagation is now possible using the skip connection (Aggarwal, 2018). The torchvision (library in PyTorch) is used to implement the model in the code. ResNet model consists of 50 convolutional layers and 1 fully connected layer. As the activation function ReLu is used after each convolutional layer. Between layers the values are normalized using batch normalization. Input convolutional layer is 7×7 with 64 channels and stride 2, followed by 3×3 max pooling layer with stride 2. After this, the layers are organized into sequences of three layers that repeat from 3 to 6 times. These sequences always have the following structure 1×1

convolutional layer, followed by 3×3 convolutional layer, followed by 1×1 convolutional layer. The number of channels varies from 64 to 2048, the highest number of channels in sequence is always in the last convolutional layer. When one sequence ends and new sequence starts, the down sampling is performed by setting the stride of first convolutional layer to 2. The model is finalized by performing average pooling and 1000-way fully connected layer (He *et al.*, 2015). The used model is pretrained as it solves the problem of data instability, as it is described in the book (Aggarwal, 2018) "One issue with using deep configurations was that increased depth led to greater sensitivity with initialization, which is known to cause instability. This problem was solved by using pretraining…"

Layers in VGG-11 are organized as shown in Table 3.1.

Table 3.1

VGG-11 Architecture (adapted from (Simonyan and Zisserman, 2015))

Section	Output size	Operation
Section 1	112×112	Convolution 7×7 at stride 2 with 64 channels
Section 2	56×56	Max pooling 3×3 at stride 2
Section 3	56×56	$\begin{bmatrix} 1 \times 1, 64\ channels \\ 3 \times 3, 64\ channels \\ 1 \times 1, 256\ channels \end{bmatrix} \times 3$
Section 4	28×28	$\begin{bmatrix} 1 \times 1, 128\ channels \\ 3 \times 3, 128\ channels \\ 1 \times 1, 512\ channels \end{bmatrix} \times 4$
Section 5	14×14	$\begin{bmatrix} 1 \times 1, 256\ channels \\ 3 \times 3, 256\ channels \\ 1 \times 1, 1024\ channels \end{bmatrix} \times 6$
Section 6	7×7	$\begin{bmatrix} 1 \times 1, 512\ channels \\ 3 \times 3, 512\ channels \\ 1 \times 1, 2048\ channels \end{bmatrix} \times 3$
Section 7	1×1	Average pooling
Section 8	1000	Fully connected

Visualization for the model is illustrated in Figure 3.5.

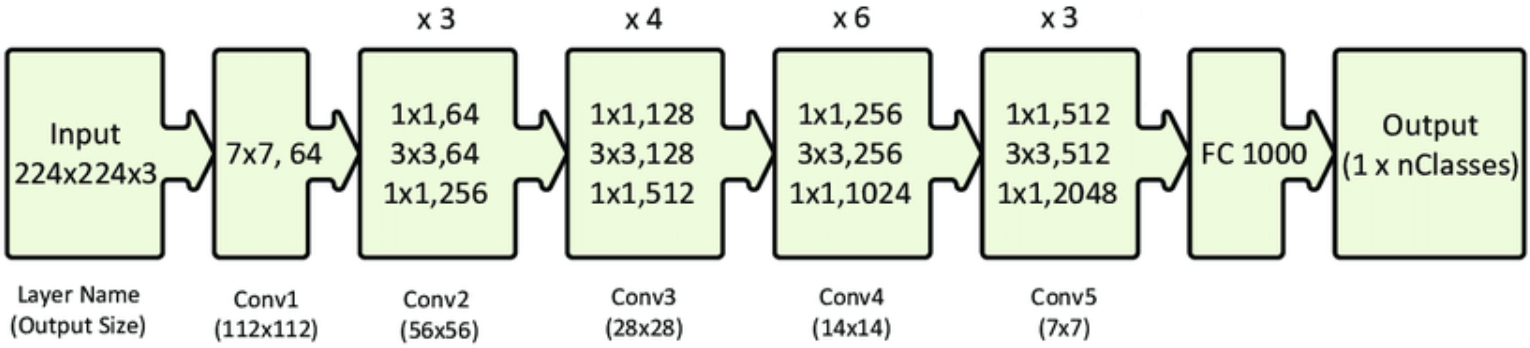

Fig. 3.5. ResNet50 architecture visualization (adapted from (Mahmood *et al.*, 2020)

3.4. Probability distribution functions

To predict the class of an image, it is needed to transform the logits in the output layer into probabilities using probability distribution function. By transforming the outputs into probabilities, it is possible to predict to which class the model classified the inputs to belong.

For the experiments, following probability distribution functions are used:

1) softmax,
2) soft-margin softmax,
3) taylor softmax.

3.4.1. Soft-margin softmax

Soft-margin softmax is an extension of softmax in which the soft-margin is introduced. This solves the hard margin constraint via the introduction of a distance margin variable m which can go through useful margins. Variable m is a positive real number or 0. If set to zero soft-margin softmax becomes identical to softmax (Liang *et al.*, 2017). It is described to reduce the distances between the instances of the same class and to enhance the discrimination between different classes (Banerjee *et al.*, 2020).

Soft-margin softmax is calculated by Formula (3.5).

$$f(x)_i = \frac{e^{x_i - m}}{e^{x_i - m} + \sum_{j \neq x_i}^{N} e^{x_j}}, \tag{3.5}$$

where $f(x)_i$ – i-th output of softmax layer;

e – Euler's number;

x_i – i-th logit;

$\sum_{j=x_i}^{N} e^{x_j}$ – sum of all logits;

m – distance margin.

3.4.2. Taylor softmax

Taylor softmax utilizes second-order Taylor series for exponential function which can be seen in Formula 3.6.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} = \sum_{n=0}^{2} \frac{x!}{n!}, \tag{3.6}$$

where x – argument;

e – Euler's number.

Taylor softmax supports more efficient way for calculations of output weights which does not depend on output size (de Brébisson and Vincent, 2016). Formula (3.7) represents calculation of Taylor softmax.

$$f(x)_i = \frac{1 + x_i + \frac{x_i^2}{2}}{\sum_{j=1}^{N} 1 + x_j + \frac{x_j^2}{2}}, \tag{3.7}$$

where $f(x)_i$ – i-th output of softmax layer;

x_i – i-th logit;

$\sum_{j=1}^{N} 1 + x_j + \frac{x_j^2}{2}$ – sum of all logits.

4. EXPERIMENTAL FINDINGS

There are 45 different experimental configurations, experiments are performed with 1e-4 learning rate and 32 batch size. Below is the table of the results (Table 4.1). Wide borders divide the groups of experiments with different loss functions in the dataset. Results are sorted by the dataset, then loss function and then probability distribution functions (PDF). Specific color coding will be used to show best results in each experimental batch.

Table 4.1

Table of the experimental results

Dataset	Loss function	PDF	Best test accuracy	Best accuracy epoch
Fashion	CCE	Softmax	0.9394	60
Fashion	CCE	SM Softmax 0.3	0.9411	61
Fashion	CCE	Taylor Softmax	0.9357	71
Fashion	KL	Softmax	0.939	62
Fashion	KL	SM Softmax 0.3	0.9382	51
Fashion	KL	Taylor Softmax	0.9384	59
Fashion	JS	Softmax	0.9362	74
Fashion	JS	SM Softmax 0.3	0.9341	69
Fashion	JS	Taylor Softmax	0.935	75
Fashion	Focal 0.5	Softmax	0.9373	68
Fashion	Focal 0.5	SM Softmax 0.3	0.9422	62
Fashion	Focal 0.5	Taylor Softmax	0.9361	49
Fashion	Focal 2	Softmax	0.9355	60
Fashion	Focal 2	SM Softmax 0.3	0.9378	66
Fashion	Focal 2	Taylor Softmax	0.9356	69
CIFAR-10	CCE	Softmax	0.8973	41
CIFAR-10	CCE	SM Softmax 0.3	0.8988	58
CIFAR-10	CCE	Taylor Softmax	0.89	49
CIFAR-10	KL	Softmax	0.8989	51
CIFAR-10	KL	SM Softmax 0.3	0.8957	44
CIFAR-10	KL	Taylor Softmax	0.8948	30

Dataset	Loss function	PDF	Best test accuracy	Best accuracy epoch
CIFAR-10	JS	Softmax	0.8862	75
CIFAR-10	JS	SM Softmax 0.3	0.8844	65
CIFAR-10	JS	Taylor Softmax	0.8917	63
CIFAR-10	Focal 0.5	Softmax	0.8942	35
CIFAR-10	Focal 0.5	SM Softmax 0.3	0.8935	32
CIFAR-10	Focal 0.5	Taylor Softmax	0.8937	61
CIFAR-10	Focal 2	Softmax	0.8907	56
CIFAR-10	Focal 2	SM Softmax 0.3	0.8902	63
CIFAR-10	Focal 2	Taylor Softmax	0.8898	38
KMNIST	CCE	Softmax	0.9687	75
KMNIST	CCE	SM Softmax 0.3	0.9639	59
KMNIST	CCE	Taylor Softmax	0.9673	58
KMNIST	KL	Softmax	0.9665	69
KMNIST	KL	SM Softmax 0.3	0.9659	57
KMNIST	KL	Taylor Softmax	0.9654	74
KMNIST	JS	Softmax	0.9564	70
KMNIST	JS	SM Softmax 0.3	0.9586	64
KMNIST	JS	Taylor Softmax	0.9603	68
KMNIST	Focal 0.5	Softmax	0.9626	41
KMNIST	Focal 0.5	SM Softmax 0.3	0.9648	70
KMNIST	Focal 0.5	Taylor Softmax	0.9655	73
KMNIST	Focal 2	Softmax	0.9666	73
KMNIST	Focal 2	SM Softmax 0.3	0.9636	60
KMNIST	Focal 2	Taylor Softmax	0.964	66

Red cells – highest accuracy in the dataset, green cells – best accuracy in a set of experiments, orange cells – lowest number of epochs before convergence in the dataset, yellow cells – lowest number of epochs before convergence in a set of experiments.

The differences observed in best accuracy are slight, however the number of epochs before convergence varies significantly. Colored cells can be counted to determine which PDF worked best overall and with specific loss functions. There are 15 sets of experiments, there are 5 sets for each dataset, each set features 3

experiments with specific loss function. Two cells are colored in each set of experiments, one for best accuracy in the set, and one for fastest convergence.

Table 4.2 features the comparison of the loss functions based on their placement in each dataset for best accuracy and best epoch. The loss functions with best accuracy and convergence speed are awarded five points, while each subsequent one is awarded one less point, until last place which gets one point. If several loss functions have similar placement in a category the number of points for the places they had to receive are divided among them equally. Each cell will store number of points awarded to loss function for specific category.

Table 4.2

Placement based loss function comparison

Dataset and metric	**CCE**	**KL**	**JS**	**Focal 0.5**	**Focal 2**
Fashion best accuracy	4	3	1	5	2
CIFAR-10 best accuracy	4	5	2	3	1
KMNIST best accuracy	5	3	1	2	4
Sum of points for accuracy	13	11	4	10	7
Fashion best epoch	2.5	4	1	5	2.5
CIFAR-10 best epoch	2	5	1	4	3
KMNIST best epoch	3	4	1	5	2
Sum of points for convergence	7.5	13	3	14	7.5
Overall sum	20.5	24	7	24	14.5

The table above can be analyzed from three perspectives, the performance of

loss function with respect to the best accuracy, fastest convergence and overall. Following conclusions can be made:

1) overall best performing functions are KL divergence and Focal loss with gamma equal to 0.5, the difference being is that KL scores higher for accuracy, while Focal 0.5 converges faster;
2) the worst performing function overall, as well as in each category is a JS divergence;
3) the best performer in accuracy is CCE with 13 points, following close are KL with 11 points and Focal 0.5 with 10 points;
4) the best performance in epochs are Focal 0.5 with 14 points and KL with 13 points.

Following table (Table 4.3) indicates how many times PDFs are featured in best runs for each loss function with respect to best accuracy, and fastest convergence rate.

Table 4.3

Loss functions performance with different PDFs

PDF and metric	**CCE**	**KL**	**JS**	**Focal 0.5**	**Focal 2**
Softmax best accuracy	1	3	1	1	2
SM Softmax best accuracy	2	0	0	1	1
Taylor Softmax best accuracy	0	0	2	1	0
Softmax best epoch	2	0	0	1	1
SM Softmax best epoch	0	2	2	1	1
Taylor Softmax best epoch	1	1	1	1	1

From the table following conclusions about performance of different configurations can be made:

1) CCE performance with best accuracy and fastest convergence in most cases happens with SM Softmax and Softmax respectively;
2) KL divergence results in best accuracy with Softmax in all cases, however, converges faster with SM Softmax;
3) JS divergence best accuracy is with Taylor Softmax, the best convergence is with SM Softmax;
4) Focal 0.5 showed equal number of best accuracy and best epochs runs for every PDF;
5) Focal 2 best accuracy happens with Softmax in most cases, however the convergence is equal for all three PDFs.

Table 4.4 shows the number of times each probability distribution function was featured in the experiment with best accuracy, or fastest convergence in the set of experiments, and is a summarized version of Table 4.3.

Table 4.4

Probability distribution functions performance comparison

Metric	Softmax	Soft-margin Softmax	Taylor Softmax
Best accuracy	8	4	3
Best epoch	4	6	5

The table indicates that the experimental configurations with Softmax resulted in best accuracy in 53.3%, while SM Softmax was best in 26.6 percent and Taylor Softmax in 20%. For best epochs the results are more balanced with SM Softmax being in the lead with 40%, Taylor Softmax with 33.3% and Softmax with 26.6 percent.

4.1. Fashion MNIST plots

In this part the accuracy plots for experiments with Fashion MNIST will be illustrated. The plots will illustrate the comparison of loss functions with same PDF used. For best experiment in each plot the confusion matrix will be shown as well. Figure 4.1 features experiments with different loss function with Softmax. The

functions by color are following:

1) CCE – orange (best accuracy = **0.9394**),
2) KL – light blue (best accuracy = 0.939),
3) JS – blue (best accuracy = 0.9362),
4) Focal 0.5 – green (best accuracy = 0.9373),
5) Focal 2 – red (best accuracy = 0.9355).

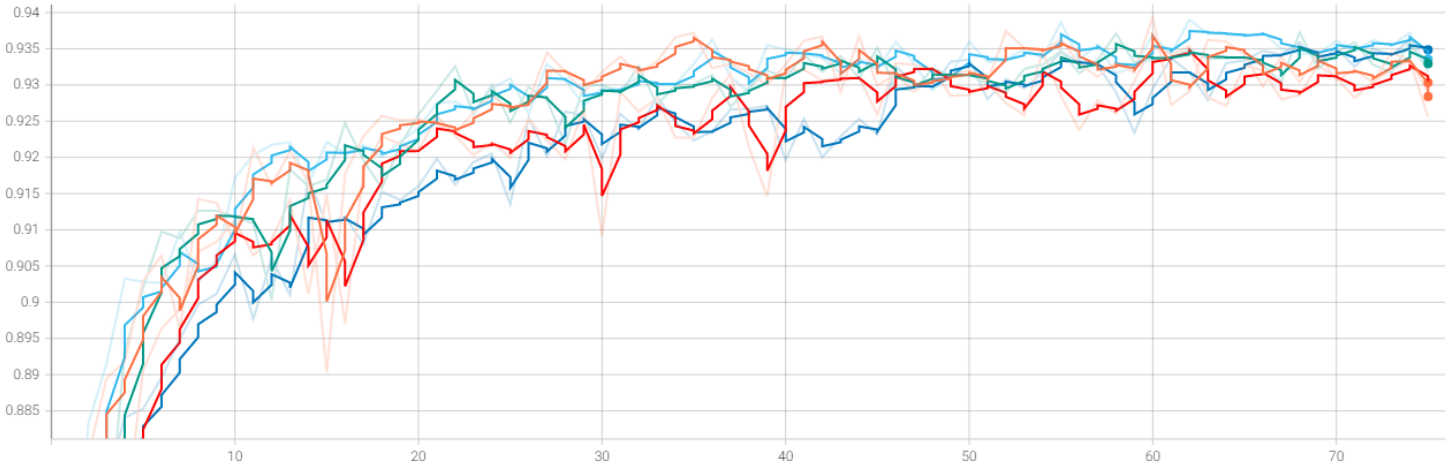

Fig. 4.1. Fashion MNIST accuracy plot for loss functions with Softmax

Figure 4.2 illustrates confusion matrix for best performing function in this batch of experiments.

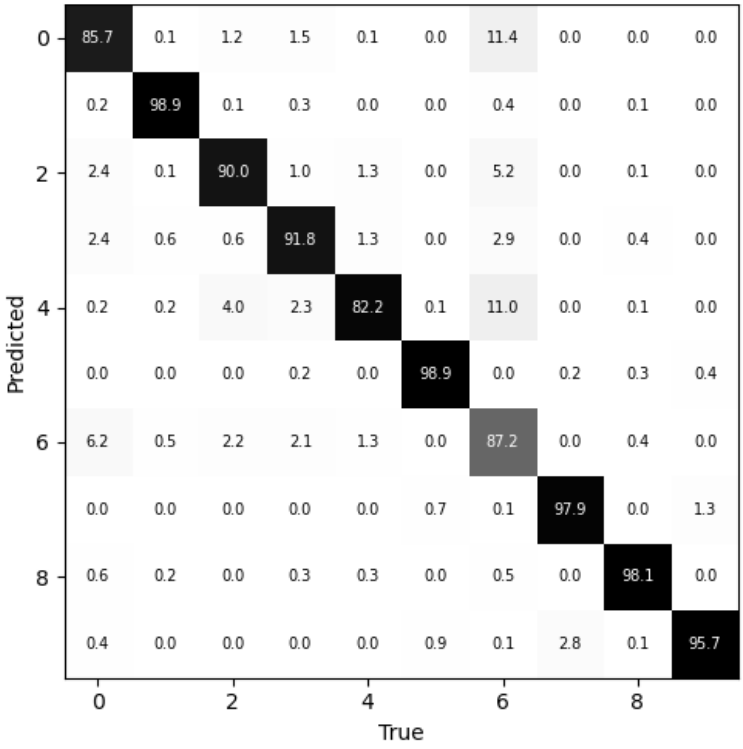

Fig. 4.2. Fashion MNIST confusion matrix for CCE with Softmax

Figure 4.3 features experiments with different loss function with SM Softmax. The functions by color are following:

1) CCE – red (best accuracy = 0.9411),
2) KL – purple (best accuracy = 0.9382),
3) JS – green (best accuracy = 0.935),
4) Focal 0.5 – grey (best accuracy = **0.9422**),
5) Focal 2 – blue (best accuracy = 0.9378).

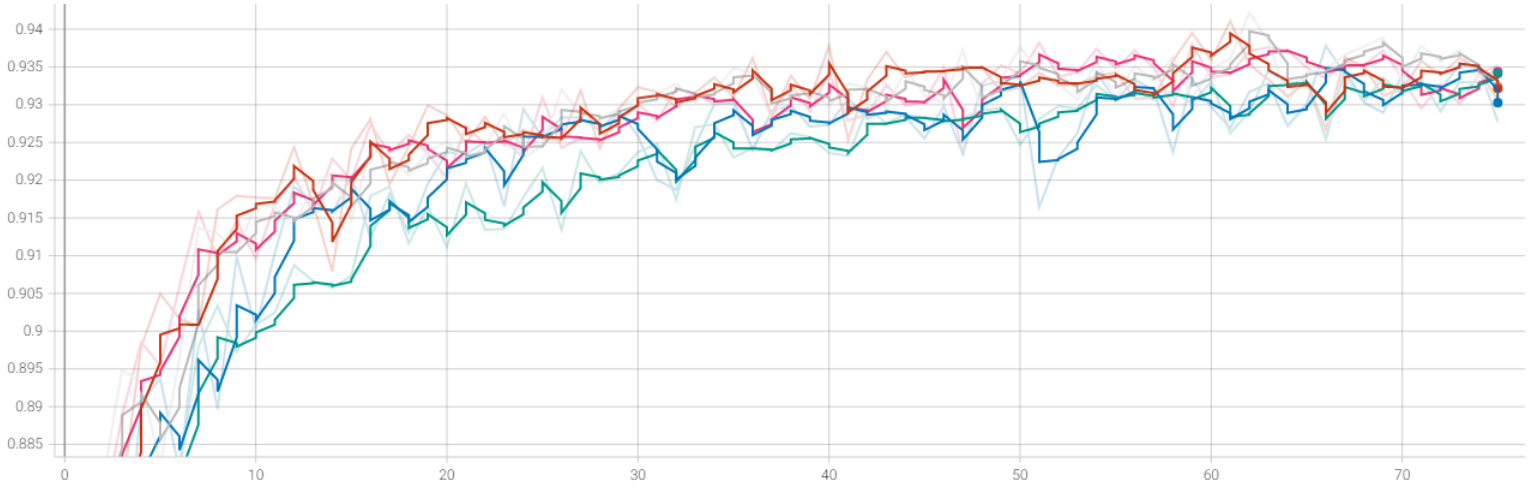

Fig. 4.3. Fashion MNIST accuracy plot for loss functions with SM Softmax

Figure 4.4 illustrates confusion matrix for best performing function in this batch of experiments.

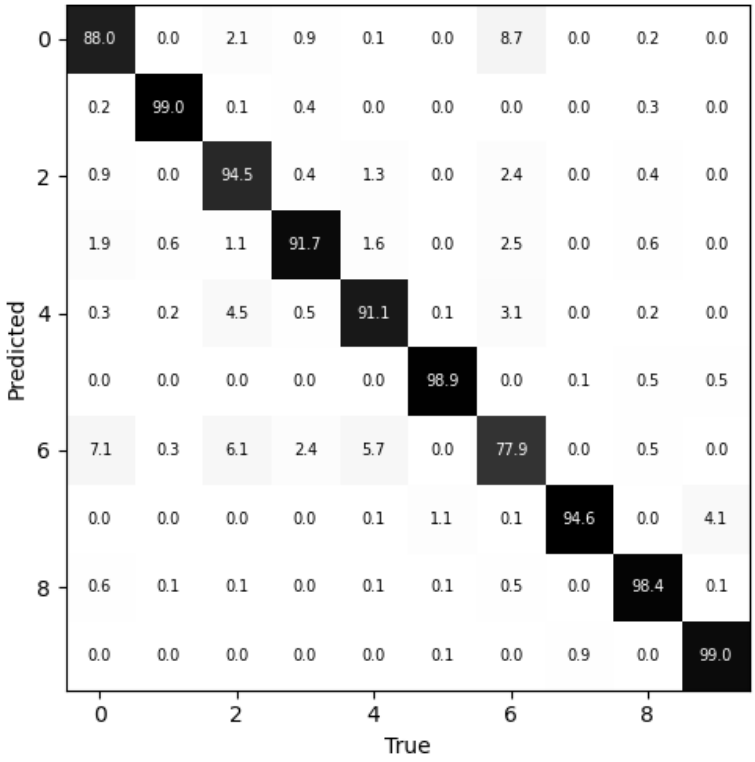

Fig. 4.4. Fashion MNIST confusion matrix for Focal 0.5 with SM Softmax

Figure 4.5 features experiments with different loss function with Taylor Softmax. The functions by color are following:

1) CCE – pink (best accuracy = 0.9357),
2) KL – grey (best accuracy = **0.9384**),
3) JS – purple (best accuracy = 0.9341),
4) Focal 0.5 – blue (best accuracy = 0.9361),
5) Focal 2 – red (best accuracy = 0.9356).

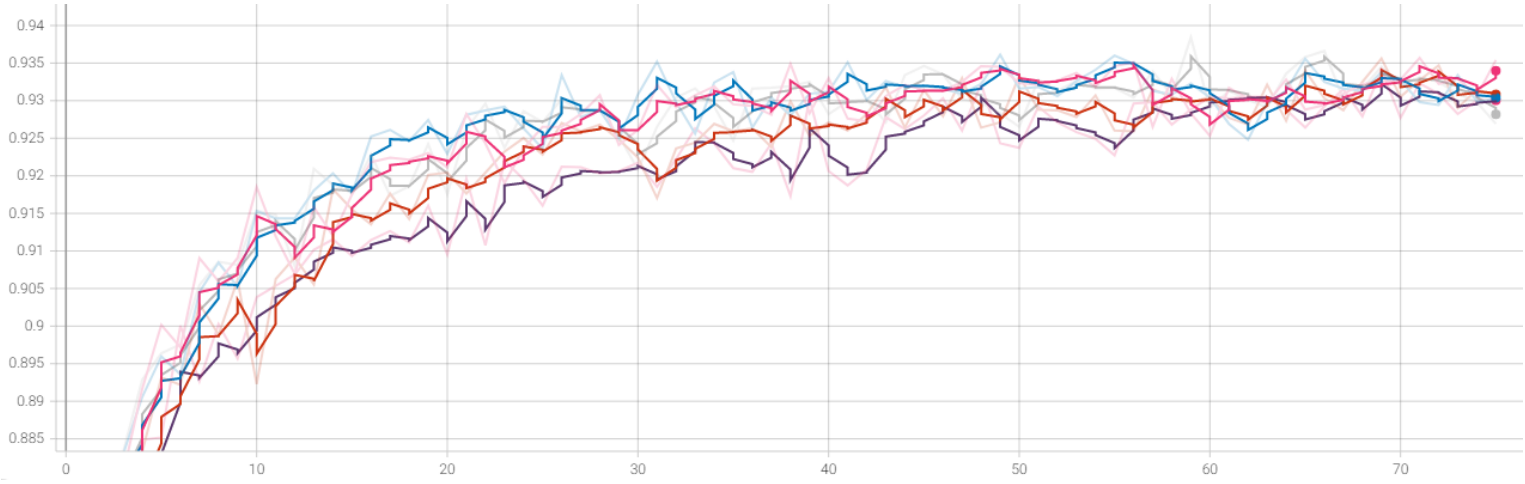

Fig. 4.5. Fashion MNIST accuracy plot for loss functions with Taylor Softmax

Figure 4.6 illustrates confusion matrix for best performing function in this batch of experiments.

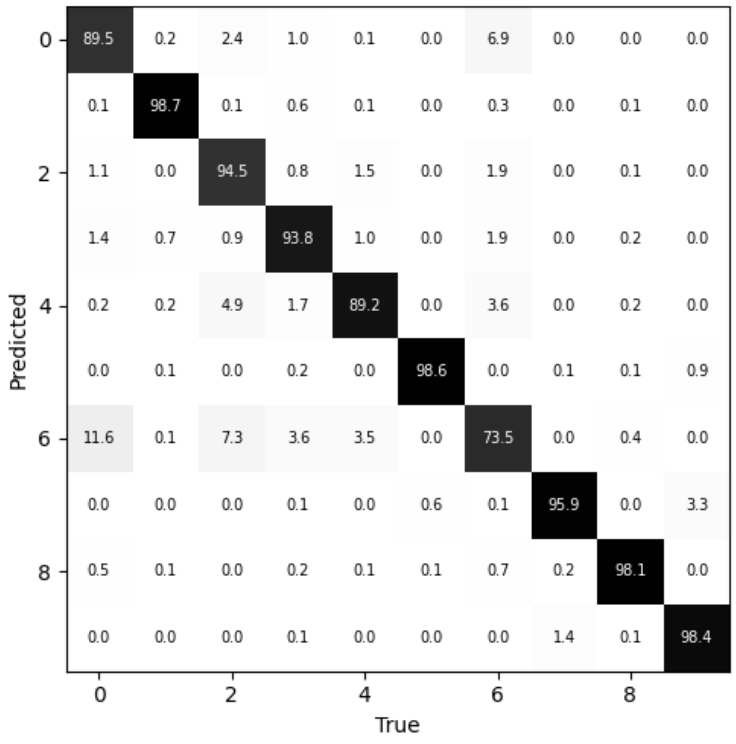

Fig. 4.6. Fashion MNIST confusion matrix for KL with Taylor Softmax

Confusion matrixes indicate that the 7th class is hardest to classify in Fashion MNIST in any configuration.

4.2. CIFAR-10 plots

This subsection follows the same structure as the previous one, however for

CIFAR-10 dataset.

Figure 4.7 features experiments with different loss function with Softmax. The functions by color are following:

1) CCE – blue (best accuracy = 0.8973),
2) KL – red (best accuracy = **0.8989**),
3) JS – purple (best accuracy = 0.8862),
4) Focal 0.5 – pink (best accuracy = 0.8942),
5) Focal 2 – grey (best accuracy = 0.8907).

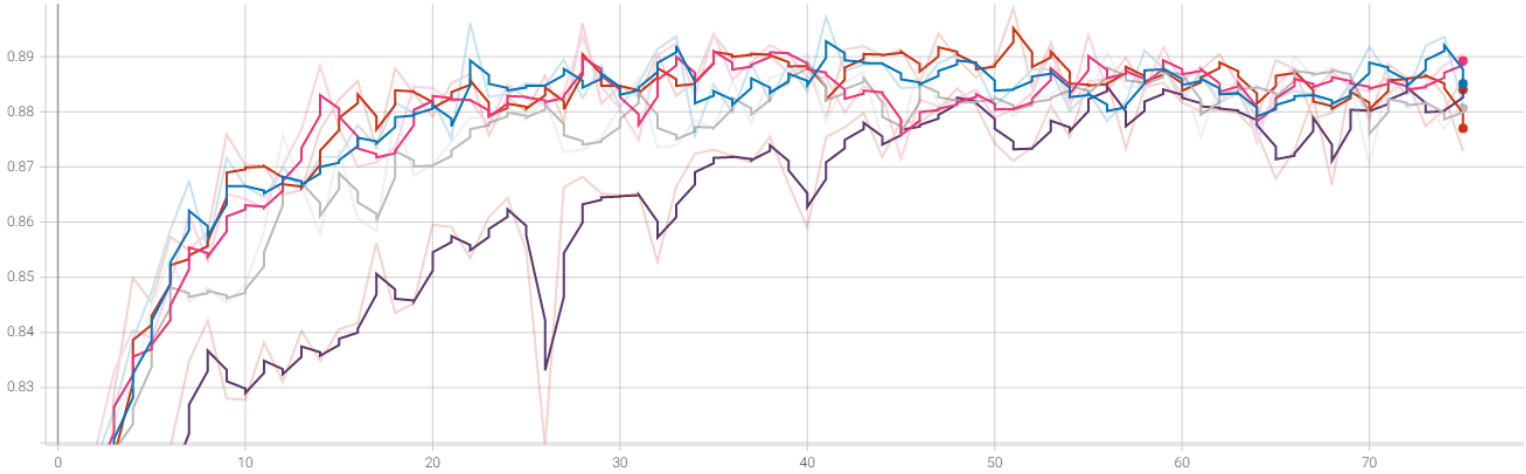

Fig. 4.7. CIFAR-10 accuracy plot for loss functions with Softmax

Figure 4.8 illustrates confusion matrix for best performing function in this batch of experiments.

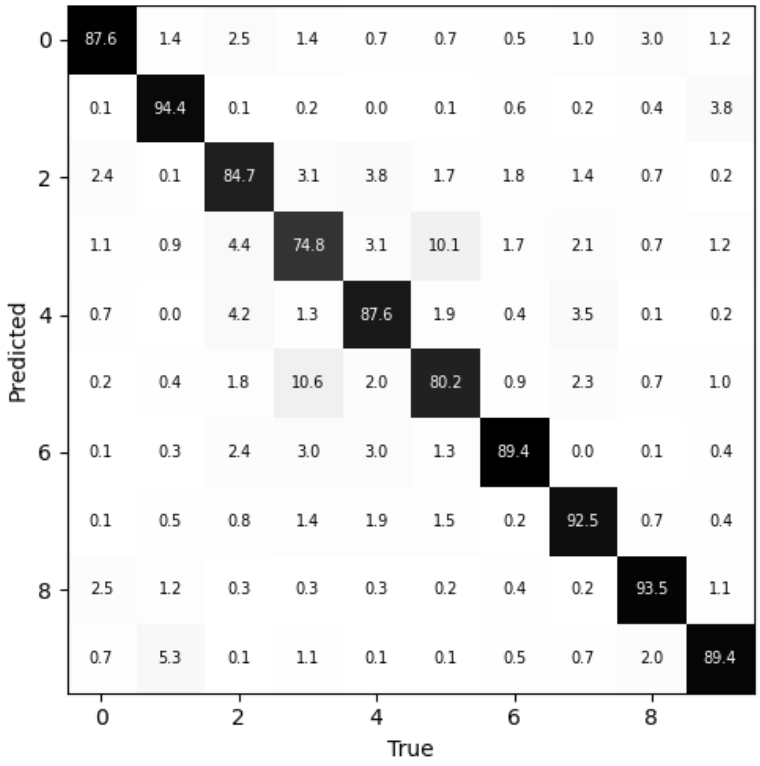

Fig. 4.8. CIFAR-10 confusion matrix for KL with Softmax

Figure 4.9 features experiments with different loss function with SM Softmax. The functions by color are following:

1) CCE – blue (best accuracy = **0.8988**),
2) KL – light blue (best accuracy = 0.8957),
3) JS – orange (best accuracy = 0.8917),
4) Focal 0.5 – green (best accuracy = 0.8935),
5) Focal 2 – purple (best accuracy = 0.8902).

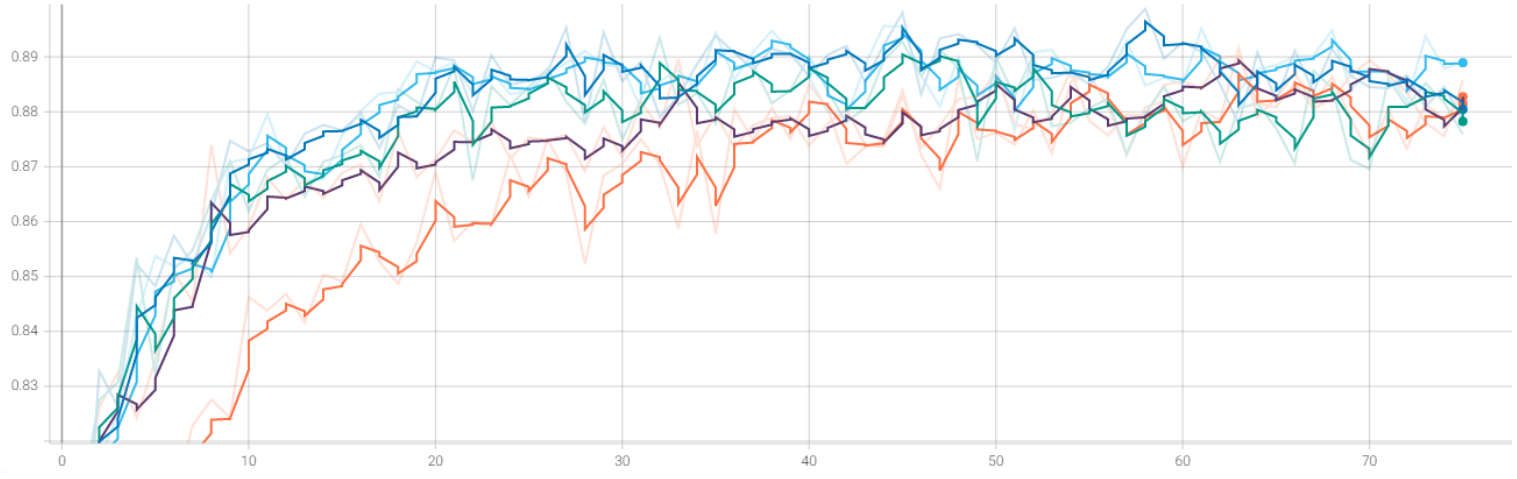

Fig. 4.9. CIFAR-10 accuracy plot for loss functions with SM Softmax

Figure 4.10 illustrates confusion matrix for best performing function in this batch of experiments.

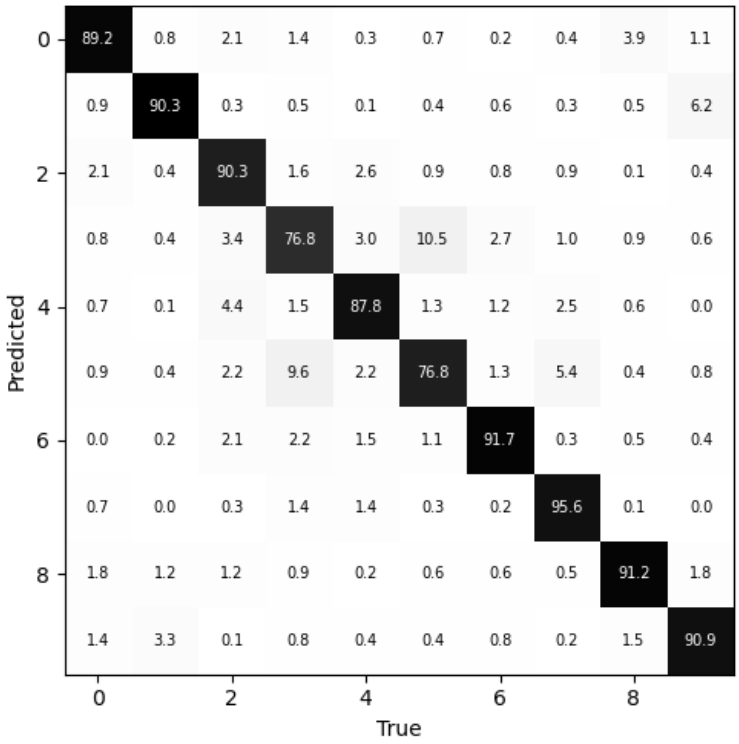

Fig. 4.10. CIFAR-10 confusion matrix for CCE with SM Softmax

Figure 4.11 features experiments with different loss function with Taylor Softmax. The functions by color are following:

1) CCE – red (best accuracy = 0.89),
2) KL – blue (best accuracy = **0.8948**),
3) JS – grey (best accuracy = 0.8844),
4) Focal 0.5 – green (best accuracy = 0.8937),
5) Focal 2 – orange (best accuracy = 0.8898).

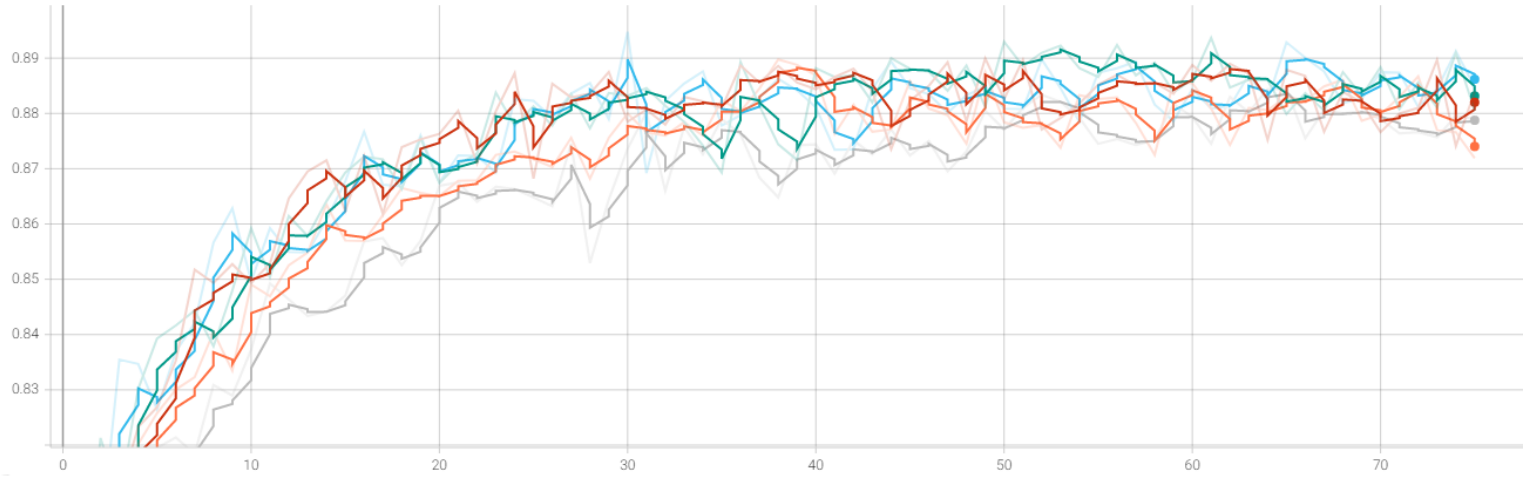

Fig. 4.11. CIFAR-10 accuracy plot for loss functions with Taylor Softmax

Figure 4.12 illustrates confusion matrix for best performing function in this batch of experiments.

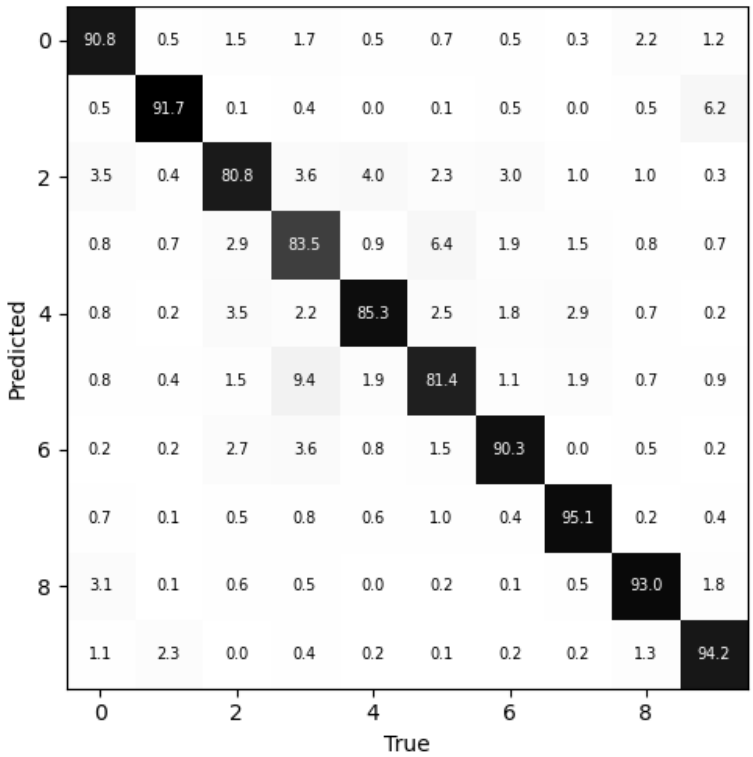

Fig. 4.12. CIFAR-10 confusion matrix for KL with Taylor Softmax

In case of CIFAR-10 4th class is the hardest to classify in first two instances, but in last confusion matrix 3rd class is hardest.

4.3. KMNIST plots

This subsection follows the same structure as the previous two, but for KMNIST dataset.

Figure 4.13 features experiments with different loss function with Softmax. The functions by color are following:

1) CCE – blue (best accuracy = **0.9687**),
2) KL – pink (best accuracy = 0.9665),
3) JS – purple (best accuracy = 0.9564),
4) Focal 0.5 – green (best accuracy = 0.9626),
5) Focal 2 – orange (best accuracy = 0.9666).

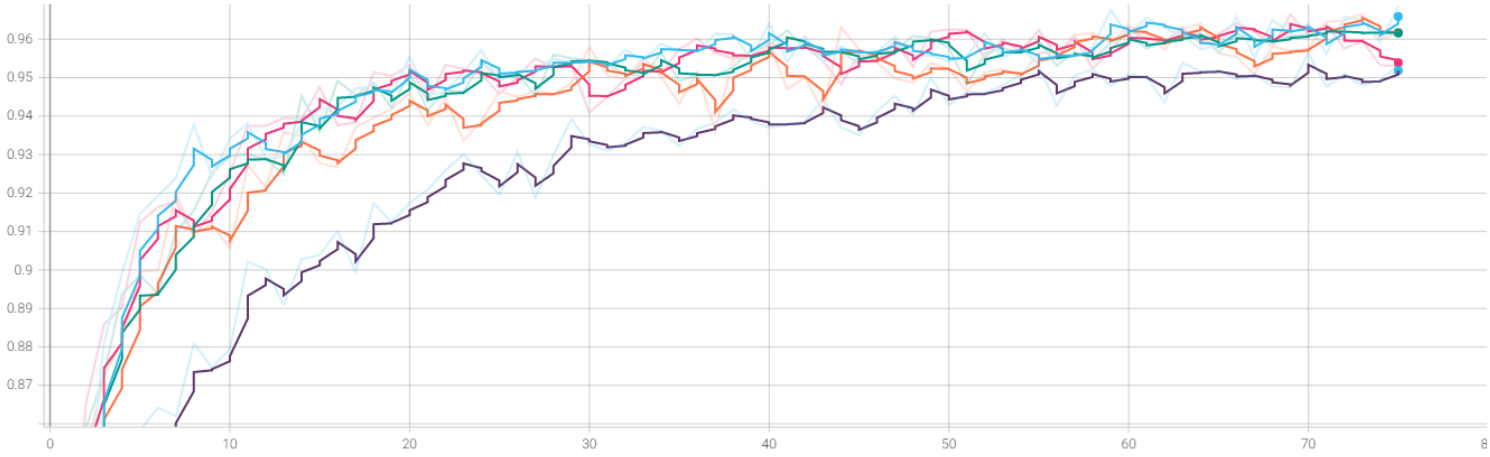

Fig. 4.13. KMNIST accuracy plot for loss functions with Softmax

Figure 4.14 illustrates confusion matrix for best performing function in this batch of experiments.

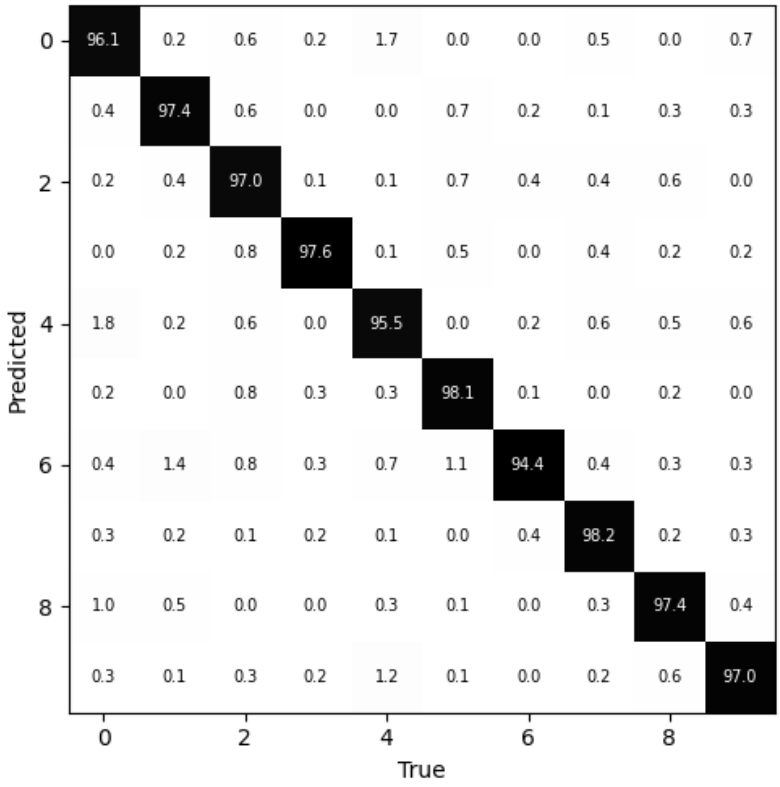

Fig. 4.14. KMNIST confusion matrix for CCE with Softmax

Figure 4.15 features experiments with different loss function with SM Softmax. The functions by color are following:

1) CCE – grey (best accuracy = 0.9639),
2) KL – blue (best accuracy = **0.9659**),
3) JS – red (best accuracy = 0.9603),
4) Focal 0.5 – light blue (best accuracy = 0.9648),
5) Focal 2 – green (best accuracy = 0.9636).

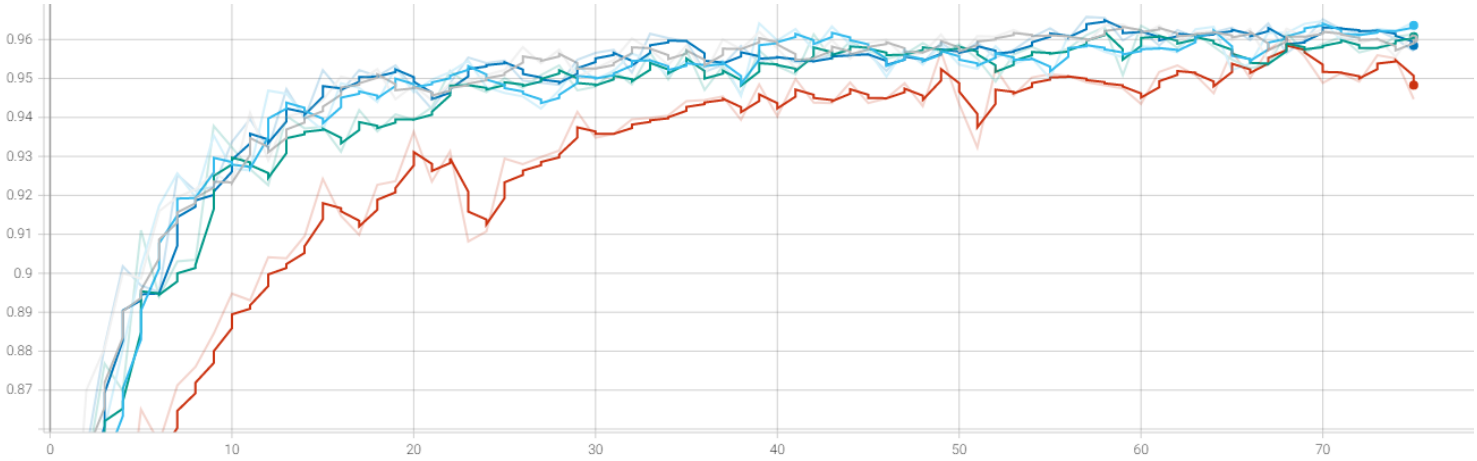

Fig. 4.15. KMNIST accuracy plot for loss functions with SM Softmax

Figure 4.16 illustrates confusion matrix for best performing function in this batch of experiments.

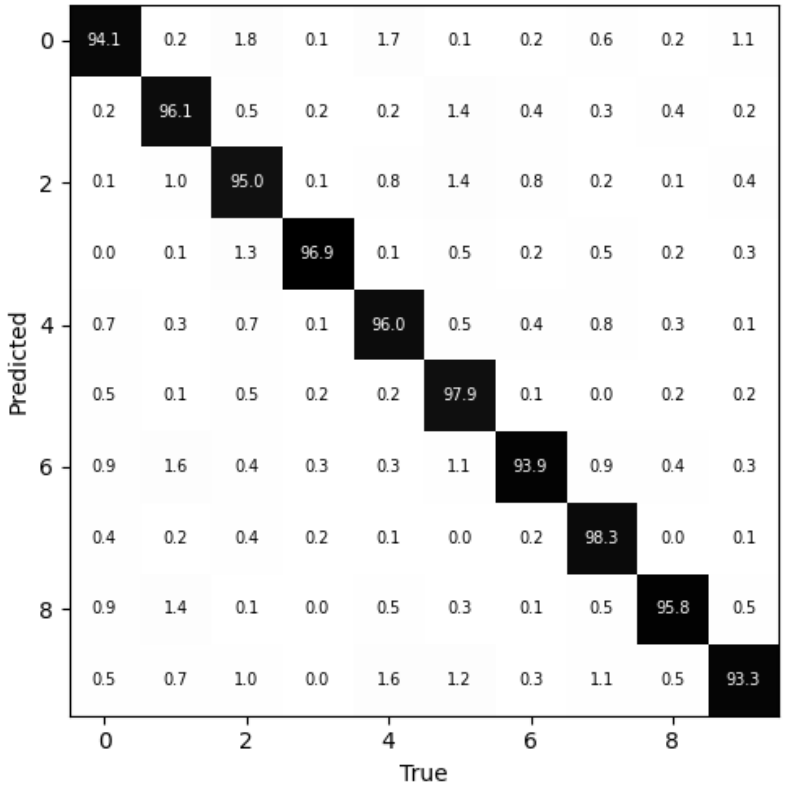

Fig. 4.16. KMNIST confusion matrix for KL with SM Softmax

Figure 4.17 features experiments with different loss function with Taylor Softmax. The functions by color are following:

1) CCE – red (best accuracy = **0.9673**),
2) KL – pink (best accuracy = 0.9654),
3) JS – blue (best accuracy = 0.9586),
4) Focal 0.5 – grey (best accuracy = 0.9655),
5) Focal 2 – orange (best accuracy = 0.964).

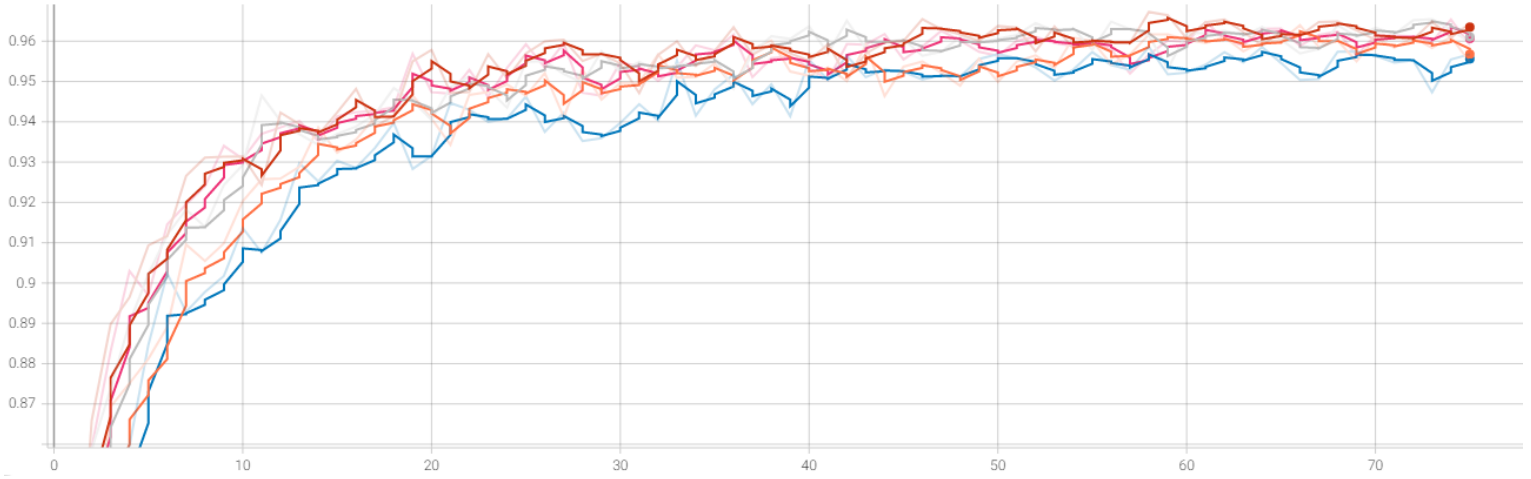

Fig. 4.17. KMNIST accuracy plot for loss functions with Taylor Softmax

Figure 4.18 illustrates confusion matrix for best performing function in this batch of experiments.

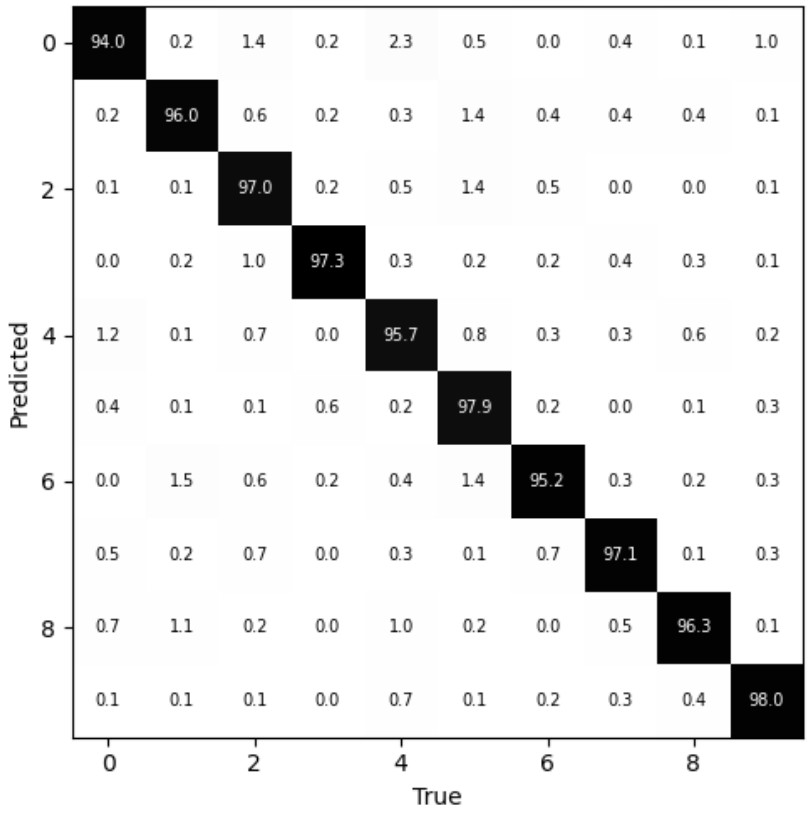

Fig. 4.18. KMNIST confusion matrix for CCE with Taylor Softmax

In case of KMNIST there are no outlier classes that are particularly hard for the model to classify.

4.4. Statistical analysis

First table (Table 4.5) indicates the t-test results for PDFs. The number can be interpreted as the probability of likeliness of source for two data sets. Data sets consist of accuracy and epoch results for runs in different datasets. They are paired, with similar loss functions being compared to each other. The probabilities are calculated using capabilities of Microsoft Excel with built-in "T.Test" function.

Table 4.5

T-Test results for different PDFs

Comparison pair	Probability for accuracy	Probability for epoch
Fashion Softmax – SM Softmax	0.19147444	0.183312067
Fashion Softmax – Taylor Softmax	0.054350476	0.485989668
Fashion SM Softmax – Taylor Softmax	0.075271874	0.266825996
CIFAR Softmax – SM Softmax	0.146289653	0.439888102
CIFAR Softmax – Taylor Softmax	0.2653766	0.361225156
CIFAR SM Softmax – Taylor Softmax	0.424281402	0.334316893
KMNIST Softmax – SM Softmax	0.298419869	0.343603045
KMNIST Softmax – Taylor Softmax	0.40212366	0.401648838
KMNIST SM Softmax – Taylor Softmax	0.080882552	0.063747987

Table 4.6 illustrates the results of calculation of P-value ($\alpha = 0.05$) with respect to each loss function. Each loss function mean is subtracted from the population mean and divided by deviation, to determine if specific loss functions scores in higher percentile on average. This is done with respect to accuracy and best epoch.

Table 4.6

P-value for loss functions

Dataset/Loss function	Accuracy p-value	Epoch p-value
Fashion CCE	0.2809	0.4856
Fashion KL	0.3120	0.1916
Fashion JS	0.1471	0.1118
Fashion Focal 0.5	0.3112	0.2898
Fashion Focal 2	0.3047	0.4314
CIFAR CCE	0.2546	0.4578
CIFAR KL	0.1766	0.2460
CIFAR JS	0.1013	0.0997
CIFAR Focal 0.5	0.3905	0.2703
CIFAR Focal 2	0.2773	0.4518
KMNIST CCE	0.2086	0.4483
KMNIST KL	0.2759	0.4301
KMNIST JS	**0.0425**	0.4001
KMNIST Focal 0.5	0.4637	0.3311
KMNIST Focal 2	0.4111	0.4451

This table shows that results are statistically insignificant and cannot prove that on average one loss function is better or worse than others in these experiments. However, p-values for best experiments indicate that they are in 100^{th} percentile. More experimentation is needed to see if specific configurations are stable and consistently produce same results. One exception, being KMNIST JS which has p-value lower than 0.05, this is the result of its poor performance, overall the p-value for experiments with JS is low and comes from negative Z-score in case of accuracy and positive Z-score in case of epochs.

CONCLUSIONS

From the empirical evidence, following conclusions can be made:

1) among best configurations there is no outlier for best accuracy, with CCE being marginally better on average and KL divergence being second best;
2) among best configurations Focal 0.5 converges fastest with KL divergence being second best;
3) JS divergence showed worst results in $\frac{5}{6}$ cases, therefore it is not recommended to use it for image-classification tasks;
4) best performance in accuracy was produced by loss functions with Softmax in most cases;
5) loss functions with SM Softmax converged fastest in marginally more cases;
6) confusion matrixes indicate that configurations have complexities with classification of similar classes;
7) the p-test and t-test showed that the results are statistically insignificant.

Even though the difference in accuracy is marginal for different configurations, the loss functions with faster convergence could significantly lessen the time needed for training of models. Configurations with fastest convergence provide up to 2.5 decrease in the number of epochs needed to converge. However, to prove this more experimentation is needed to see if these results can be consistently achieved.

FURTHER RESEARCH

For further research, more diverse datasets are advised. The datasets used in this research were all filled with low resolution images that were divided into 10 classes. The experiments with datasets that have more features per image, as well as more possible classes could unveil new patterns and produce more statistically significant results.

Focal loss can be analyzed further by trying other gamma values, 0.5 and 2 were used in this research. Same applies to SM Softmax, in this research only 0.3 margin was used for experimentation.

Comparison of novel loss functions like the ones introduced in (Berlyand, Creese and Jabin, 2021) and (Wang and Yang, 2019) can be performed to widen the array of possibilities for choice of loss function in image-classification tasks in case of satisfactory results.

The correlation between the choice of loss function with specific PDF with the speed of convergence should be researched more to further improve the training speed of models.

LIST OF REFERENCES

Aggarwal, C.C. (2018) *Neural networks and deep learning: a textbook*. Cham, Switzerland: Springer. doi:10.1007/978-3-319-94463-0.

Banerjee, K. *et al.* (2020) 'Exploring Alternatives to Softmax Function', *arXiv:2011.11538 [cs]* [Preprint]. Available at: http://arxiv.org/abs/2011.11538 (Accessed: 24 April 2022).

Berlyand, L., Creese, R. and Jabin, P.-E. (2021) 'A novel multi-scale loss function for classification problems in machine learning'. arXiv. Available at: http://arxiv.org/abs/2106.02676 (Accessed: 28 May 2022).

de Brébisson, A. and Vincent, P. (2016) 'An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family', *arXiv:1511.05042 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1511.05042 (Accessed: 24 April 2022).

Buduma, N. and Locascio, N. (2017) *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*. First edition. Sebastopol, CA: O'Reilly Media.

Charniak, E. (2018) *Introduction to deep learning*. Cambridge, Massachusetts: The MIT Press.

Cho, K. *et al.* (2019) 'A Performance Comparison of Loss Functions', in *2019 International Conference on Information and Communication Technology Convergence (ICTC). 2019 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, Korea (South): IEEE, pp. 1146–1151. doi:10.1109/ICTC46691.2019.8939902.

Chollet, F. (2015) *Keras*. GitHub. Available at: https://github.com/fchollet/keras.

Chollet, F. (2018) 'Deep Learning with Python', in.

Clanuwat, T. *et al.* (2018) 'Deep Learning for Classical Japanese Literature'. doi:10.20676/00000341.

Daniel Zhang *et al.* (2022) *The AI Index 2022 Annual Report*. AI Index Steering Committee, Stanford Institute for Human-Centered AI, Stanford University.

Fuglede, B. and Topsoe, F. (2004) 'Jensen-Shannon divergence and Hilbert space embedding', in *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings. International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, Chicago, Illinois, USA: IEEE, pp. 30–30. doi:10.1109/ISIT.2004.1365067.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep learning*. Cambridge, Massachusetts: The MIT Press (Adaptive computation and machine learning).

He, K. *et al.* (2015) 'Deep Residual Learning for Image Recognition'. arXiv. Available at: http://arxiv.org/abs/1512.03385 (Accessed: 24 May 2022).

Kingma, D.P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization'. arXiv. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 24 May 2022).

Krizhevsky, A. (2009) 'Learning Multiple Layers of Features from Tiny Images', in.

Krizhevsky, A., Nair, V. and Hinton, G. (2009) 'CIFAR-10 (Canadian Institute for Advanced Research)'. Available at: http://www.cs.toronto.edu/~kriz/cifar.html.

Liang, X. *et al.* (2017) 'Soft-Margin Softmax for Deep Classification', in Liu, D. et al. (eds) *Neural Information Processing*. Cham: Springer International Publishing (Lecture Notes in Computer Science), pp. 413–421. doi:10.1007/978-3-319-70096-0_43.

Lin, T.-Y. *et al.* (2018) 'Focal Loss for Dense Object Detection', *arXiv:1708.02002 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1708.02002 (Accessed: 24 April 2022).

Liu, L. (2018) *Encyclopedia of database systems*. New York, NY: Springer Berlin Heidelberg.

Mahmood, A. *et al.* (2020) 'Automatic Hierarchical Classification of Kelps Using Deep Residual Features', *Sensors*, 20(2), p. 447. doi:10.3390/s20020447.

Murphy, K.P. (2012) *Machine learning: a probabilistic perspective*. Cambridge, MA: MIT Press (Adaptive computation and machine learning series).

Nielsen, M.A. (2015) *Neural networks and deep learning*. Determination press San Francisco, CA, USA.

Raymond Perrault *et al.* (2019) *The AI Index 2019 Annual Report*. Stanford, CA: AI Index Steering Committee, Human-Centered AI Institute, Stanford University.

Russell, S.J. and Norvig, P. (2021) *Artificial intelligence: a modern approach*. Fourth edition. Hoboken: Pearson (Pearson series in artificial intelligence).

Sammut, C. and Webb, G.I. (eds) (2017) *Encyclopedia of machine learning and data mining*. Second edition. New York, NY: Springer (Springer reference).

Simonyan, K. and Zisserman, A. (2015) 'Very Deep Convolutional Networks for Large-Scale Image Recognition', *arXiv:1409.1556 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1409.1556 (Accessed: 22 April 2022).

Sokolova, M., Japkowicz, N. and Szpakowicz, S. (2006) 'Beyond Accuracy, F-Score

and ROC: A Family of Discriminant Measures for Performance Evaluation', in Sattar, A. and Kang, B. (eds) *AI 2006: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 1015–1021. doi:10.1007/11941439_114.

Srivastava, Y., Murali, V. and Dubey, S.R. (2019) 'A Performance Comparison of Loss Functions for Deep Face Recognition'. arXiv. Available at: http://arxiv.org/abs/1901.05903 (Accessed: 20 May 2022).

Trask, A.W. (2019) *Grokking deep learning*. Shelter Island: Manning.

Vasilev, I. *et al.* (2019) *Python deep learning: exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*. Second edition. Birmingham Mumbai: Packt Publishing Limited.

Wang, Y. and Yang, L. (2019) 'A robust loss function for classification with imbalanced datasets', *Neurocomputing*, 331, pp. 40–49. doi:10.1016/j.neucom.2018.11.024.

Xiao, H., Rasul, K. and Vollgraf, R. (2017) 'Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms', *arXiv:1708.07747 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1708.07747 (Accessed: 21 April 2022).

Yoav Shoham *et al.* (2018) *The AI Index 2018 Annual Report*. Stanford, CA: AI Index Steering Committee, Human-Centered AI Initiative, Stanford University.

Zhang, A. *et al.* (2021) 'Dive into Deep Learning', *ArXiv*, abs/2106.11342.

Zhu, H. and Kaneko, T. (2018) 'Comparison of Loss Functions for Training of Deep Neural Networks in Shogi', in *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI). 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, Taichung: IEEE, pp. 18–23. doi:10.1109/TAAI.2018.00014.