Technical Sciences, 2022, 25, 149–170

UNIWERSYTET WARMIŃSKO-MAZURSKI W OLSZTYNIE

Yearbook peer-reviewed scientific journal
ISSN 1505-4675
e-ISSN 2083-4527
TECHNICAL SCIENCES
Homepage: https://czasopisma.uwm.edu.pl/index.php/ts/

DOI: https://doi.org/10.31648/ts.8098

MULTICLASS VOICE COMMANDS CLASSIFICATION WITH MULTIPLE BINARY CONVOLUTION NEURAL NETWORKS

Jarosław Szkoła

ORCID: 0000-0002-6043-3313
Institute of Computer Sciences
University of Rzeszow

Received 17 August 2022; accepted 27 October 2022; available online 3 November 2022.

K e y w o r d s: multiclass convolution neural networks, voting decision mechanism, voice commands classification, multiclass classifier, sound wave processing and classification.

A b s t r a c t

In machine learning, in order to obtain good models, it is necessary to train the network on a large data set. It is very often a long process, and any changes to the input dataset require re-training the entire network. If the model is extended with new decision classes, the entire learning process for all samples must be repeated. To improve this process, a new neural network architecture was proposed that uses a combination of multiple smaller independent convolutional neural networks (O'SHEA, NASH 2015, ZEGHIDOUR et al. 2019) with two outputs, and a voting mechanism (CORNELIO et al. 2021, DONINI et al. 2018) that ultimately determines the response of the network decision, rather than one large single network. The main purpose of using such an architecture is the need to solve the problem that occur in the case of most multiclass neural networks. For a typical neural network, extending with new decision classes requires changing the network architecture and re-learning the model for all data. In the proposed architecture, adding a new decision class requires only adding a small independent neural network, and the learning process applies to new cases with small subset of original dataset. This architecture is proposed for large datasets with many decision classes.

Correspondence: Jarosław Szkoła, Instytut Informatyki, Zakład Metod Przybliżonych, Uniwersytet Rzeszowski, ul. Pigonia 1, 35-310 Rzeszów, e-mail: jszkola@ur.edu.pl.

Introduction

Machine learning is one of the fastest growing technologies of artificial intelligence. It means the increase in computing capabilities of modern computers and access to open databases made it possible to obtain very good results for the newly developed algorithms. Many of the databases have become the standard when it comes to the reference value for comparing different training algorithms, it is worth mentioning the most popular ones here: Boston Housing Dataset, Iris Data Set, The Mnist Database, ImageNet, HAM10000, A public dataset for single-word speech recognition. One of the most important factors determining the possibility of creating an effective model is the quality of the training data. The amount of available input data, broken down into appropriate decision classes, is of great importance. Another equally important factor is the uniformity of the distribution of training samples for individual classes in such a way, and the training set was representative. In practice, especially in the case of medical samples, it is very difficult to obtain an appropriate level of balance. Much of the open databases is unbalanced in most cases. To solve this problem, various techniques are used to preprocess the original data, starting with removing some data from the redundant class, or filling in the missing data by appropriately modifying the already existing samples. In the case of an image, a commonly known and frequently used technique is the augmentation method, which consists in creating additional images on the basis of existing ones, by simply modifying them, i.e. rotating, moving, reflecting, isometric transformations, but also, although less frequently, color saturation modifications are used, histogram alignment and others. Just as important as balancing the training data is the ability to use the right number of input samples. The quantity and quality of input samples is a key element that allows the data model to be properly trained. It is commonly believed that the more input data used in the model training stage, the better. Contrary to appearances, too much training data may deteriorate the quality of the model. This problem was noticed when for language models known as GPT-2 and GPT-3 was born. It has been shown that the much smaller "Chinchilla" model with 70 billion parameters will achieve much better results than many more complex models with much more parameters, egLaMDA: 137 Billion, GPT-3: 175 Billion, Gopher: 280 Billion (HOFFMANN et al. 2022). The selection of training samples must be closely matched to the model architecture. This can be achieved in two ways, either by designing the neural network architecture to which we will adjust the training data, or by adjusting the network architecture to the data. In the case of models whose task is to classify data, in the vast majority of solutions, at the learning stage we use data with a strictly defined number of decision classes, e.g. CIFAR-10, CIFAR-100, with 10 and 100 decision classes, respectively. However, there are models which, due to the type of data, will most likely be extended by a new class in the future,

for example a database for recognizing the words of a given language (WARDEN 2018). In most cases, in the case of extending data with new classes, it is done by modifying the existing model architecture, and then the training process for all data is carried out. There are techniques called transfer learning (SHAFAHI et al. 2020) that allow the use of existing models to classify new datasets but from the same domain. Usually this is a narrowing solution, the original large models are used as the initial parameter initializer of the smaller model, with fewer decision classes. An important advantage of this approach is the use of an already trained model to adapt to a new problem, rather than carrying out learning from the beginning, with randomly set weights. Transfer learning does not solve the problem of extending an already existing model with new decision classes that usually require training on the basis of new data samples, but in such a way that the previous knowledge encoded in the model is not lost. The problem with extending already trained models with new decision classes is not heavily explored at the moment, which in the observed large increase in new data sets becomes a significant problem.

Dataset description

A public dataset for single-word speech recognition (WARDEN 2017) was used as the training dataset, which contained 64,721 speech samples that were divided into 30 classes. The database is available at: http://download.tensorflow.org/data/speech_commands_v0.01.tar. Each class specifies one command to which an audio sample has been assigned. Each sound sample is in 16-bit little-endian PCM-encoded WAV (Waveform Audio File Format) file at sample rate 16 kHz mono. The audio was trimmed to a one second length to align most utterances. The number of available samples for each class is presented in Table 1.

Table 1

Dataset samples per command

Id	Command	Total samples per command
1	2	3
1	four	2,372
2	two	2,373
3	wow	1,745
4	happy	1,742
5	tree	1,733
6	no	2,375
7	bed	1,713
8	off	2,357
9	bird	1,731

cont. Table 1

1	2	3
10	nine	2,364
11	five	2,357
12	eight	2,352
13	zero	2,376
14	house	1,750
15	go	2,372
16	stop	2,380
17	up	2,375
18	down	2,359
19	seven	2,377
20	one	2,370
21	right	2,367
22	dog	1,746
23	left	2,353
24	on	2,367
25	six	2,369
26	three	2,356
27	cat	1,733
28	sheila	1,734
29	yes	2,377
30	marvin	1,746

Pre-processing of the dataset

The input files are saved as samples in wav format, they are often referred to as waveforms. Waveform are a time series with the amplitude of the signal at any given time. In order to obtain more information about signal, in addition to changes over time, one can also take into account the frequency distribution over time. The frequency domain representation of the signal tells us what different frequencies are present in the signal. The Fourier transform is a mathematical concept that transforms a continuous waveform from the time domain to the frequency domain (Fig. 1). An audio signal is a complex signal made up of many fundamental frequencies that propagate through the air as pressure changes. When the sound is recorded, we record the amplitude composite of these individual waves. A Fourier transform can decompose a signal into its component frequencies, and in addition to information about the component frequencies, we obtain information about the amplitude of each component. In practice, the Fast Fourier Transform (FFT) is used, which is used to process discrete waveforms. For the function $f(t)$ the Fourier transform is given by the formula 1:

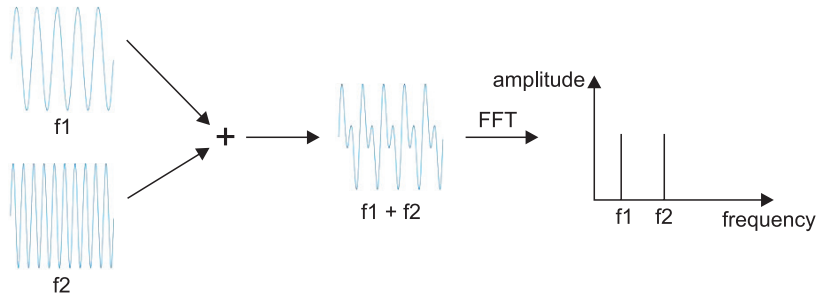

Fig. 1. Fourier transform signal processing from time domain to frequency domain

$$\hat{f}(\delta) = \int_{-\infty}^{+\infty} f(t)e^{-2\pi it\delta}dt, \text{ where } i \text{ is imaginary entity } (i^2 = -1),\ \delta = \text{frequency} \quad (1)$$

When we apply the Fourier transform to the input signal, we only get the frequency values and we lose the time information. It will therefore not be possible to recognize the sequence of waveform changes if we use these frequencies as a function. Another way to calculate the characteristics for our signal should be used, so that it has the frequency values along with the time in which the changes were observed. This is where spectrograms come into play. In the spectrogram graph, horizontal axis represents time, vertical axis represents frequencies, and the colors represent the amplitude of the processed frequency over time. An example of wave sample and spectrogram for one of the word "no" is presented in Figures 2 and 3.

For calculations and data preprocessing the TensorFlow and Keras libraries in version 2.6.0 were used. In the first step, all samples were converted to the spectral form, and next the spectrograms were adapted to the TFRecord structure. The TFRecord format is a simple format for storing a sequence of binary records. This structure is efficiency method for storing data, also is very effective for process data with CPU/GPU, because Tensorflow framework can read this data with parallel I/O operations. All steps for processing data for the neural network are presented on this Figure 4.

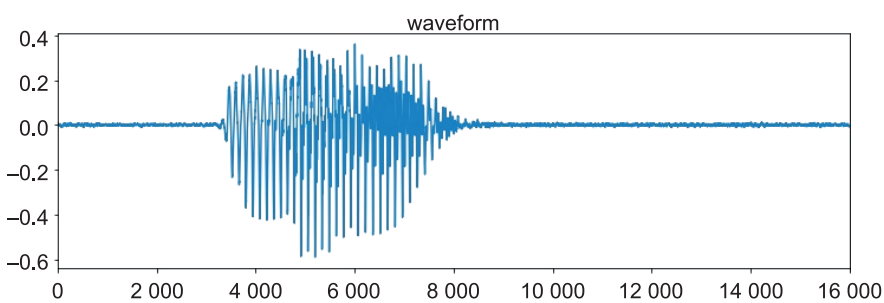

Fig. 2. Sample wave for command "no"

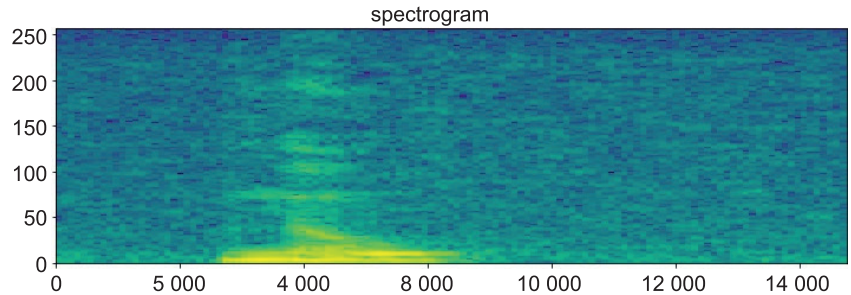

Fig. 3. Spectrogram for sample wave for command "no"

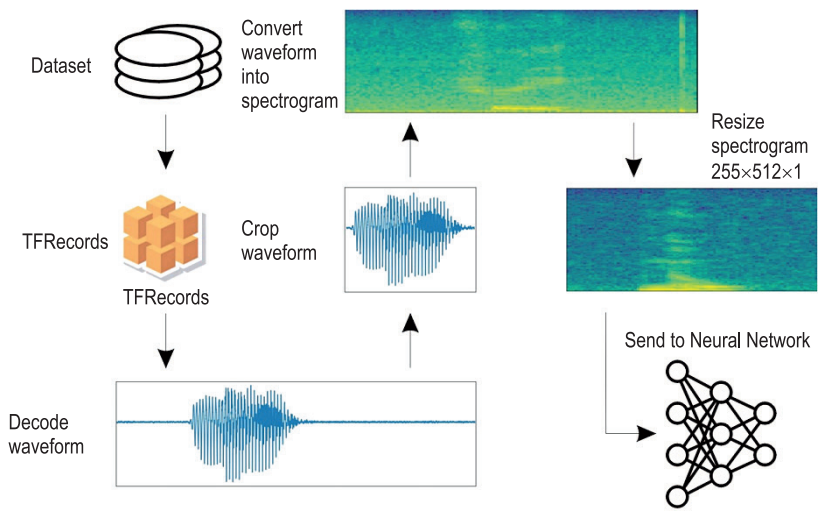

Fig. 4. All steps for pre-processing dataset for the neural network

Description of model

To check how effective the new neural network architecture is, one large multilayer convolutional network is used for testing, which is defined as the base model. In the next step, we use this same base model, but we will check the impact of reducing the input file size for the same network on classification. The original test set will be randomly split into 1/2, 1/4, 1/8, 1/16, 1/32, 1/64 of original dataset. This trained models are called benchmarks, where, before the appropriate reduction of the input sets, we want to check how small the input set can be, so that the classification level is still high (above 75%). In the last step, the modified network architecture and details of the training algorithm will be presented, and the method of determining the network responses in terms of classification.

Baseline model

As the base model, a multilayer deep convolution network was used with 20 layers and 30 outputs, of which only one of them can be active as a result of the classification. For balance the data between certain layers, technique called batch normalization was used. Neural network architecture was presented in Figure 5.

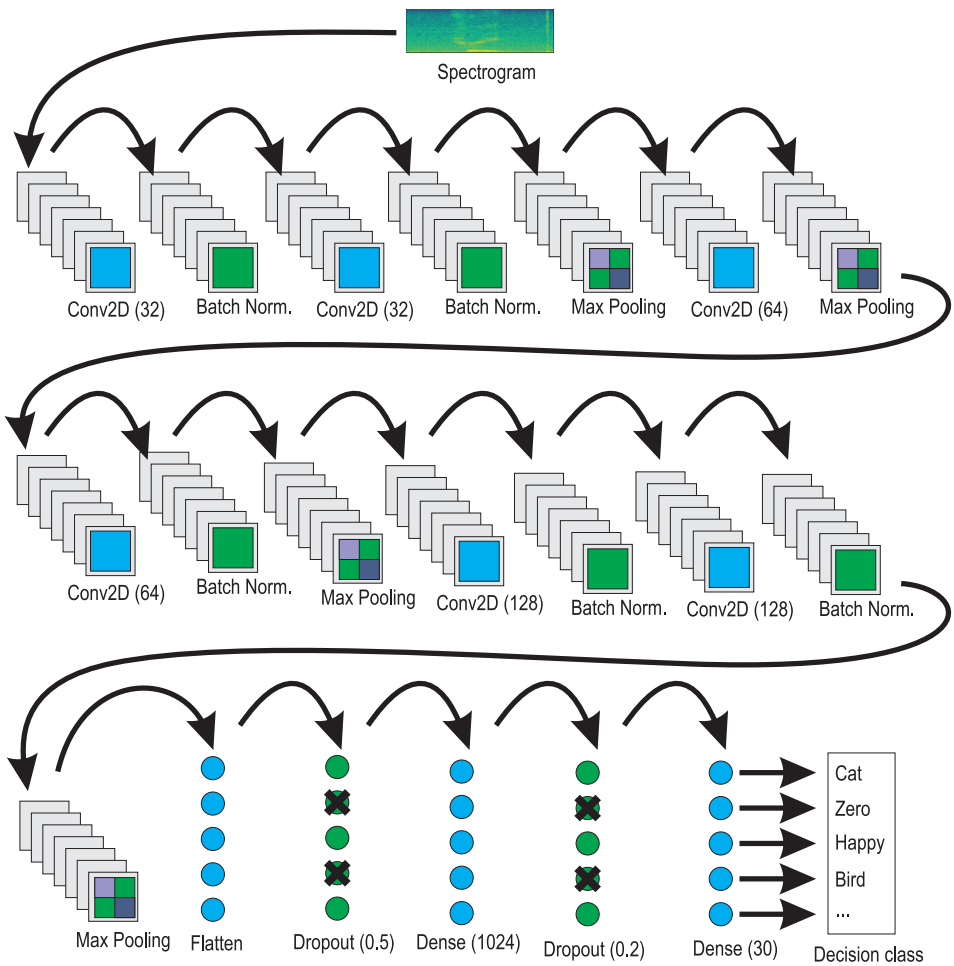

Fig. 5. Baseline model architecture

Training baseline model

The entire data set was used to prepare the reference model. The input data was randomly divided into the learning, validation and testing parts. The test set consists of 80% of randomly selected samples, the remaining 20% of the samples are divided into the validation part and the test part, 10% each. The learning process runs smoothly, and the network, after a small number of iterations, obtains very good classification results for the validation set, as shown in Figure 6. For the test data set, the model achieves a accuracy of 95%.

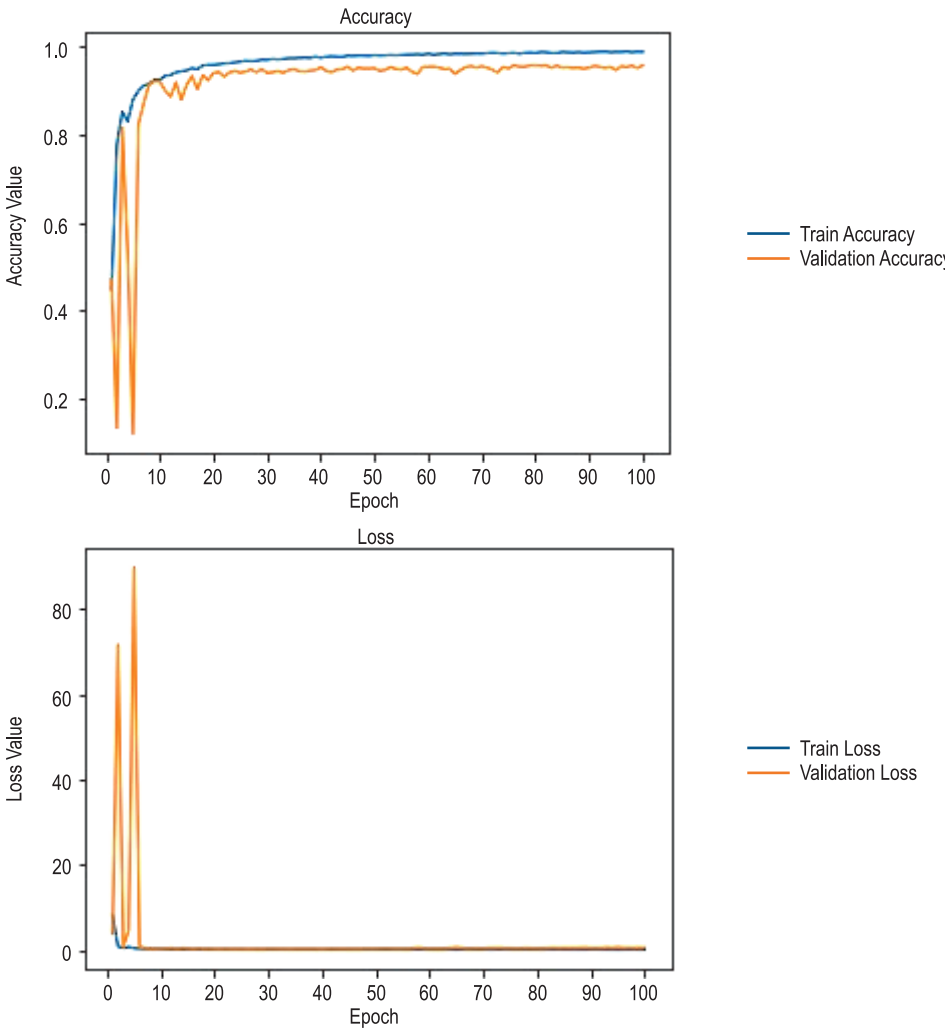

Fig. 6. Baseline model learning charts

For the given dataset, the accuracy ranged from 82.7 to 89.7% in various studies, depending on the models (WARDEN 2018). For presented model the accuracy of prediction for individual words is presented using the confusion matrix shown in Figures 7.

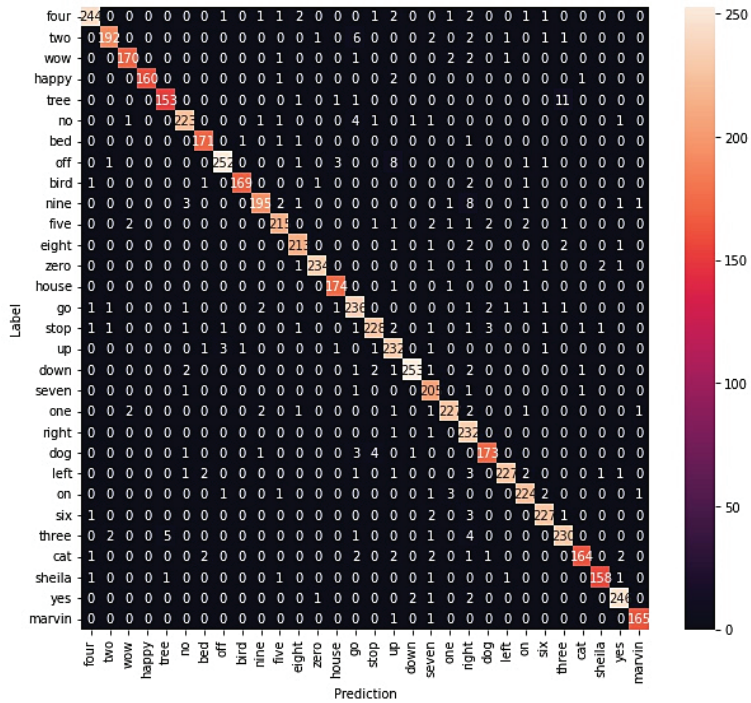

Fig. 7. Confusion matrix for baseline model

Benchmark model

In this procedure, we check how much data is needed to obtain a model with at least 75% quality. In the case of training neural networks with gradient methods, apart from the problems related to the disappearing or exploding gradient, we often have a problem with under-training or overfitting the network. If the network architecture is appropriate, the most common problem is poor quality or quantity of input data. Usually, we have too little input data divided into different classes, or within individual classes, we have too large disproportions between the number of elements between classes. The original data set was split randomly into 1/2, 1/4, 1/8, 1/16, 1/32, 1/64 of subsets. For each subset, the input data was randomly divided into the learning, validation and testing

parts. The test set consists of 80% of randomly selected samples, the remaining 20% of the samples are divided into the validation part and the test part, 10% each. After training the model for different subsets, we obtained classification level, presented in Table 2. The comparison of the learning process for the subset 1/2 and the subset 1/64 is shown in Figures 8 and 9. As can be seen in the graph, when the network does not receive enough appropriate input data, the learning process is chaotic, the loss function for the validation set grows instead of falling. In the case of the assumed threshold of 75%, the smallest subset that meets the above assumptions is 1/8 of the original set.

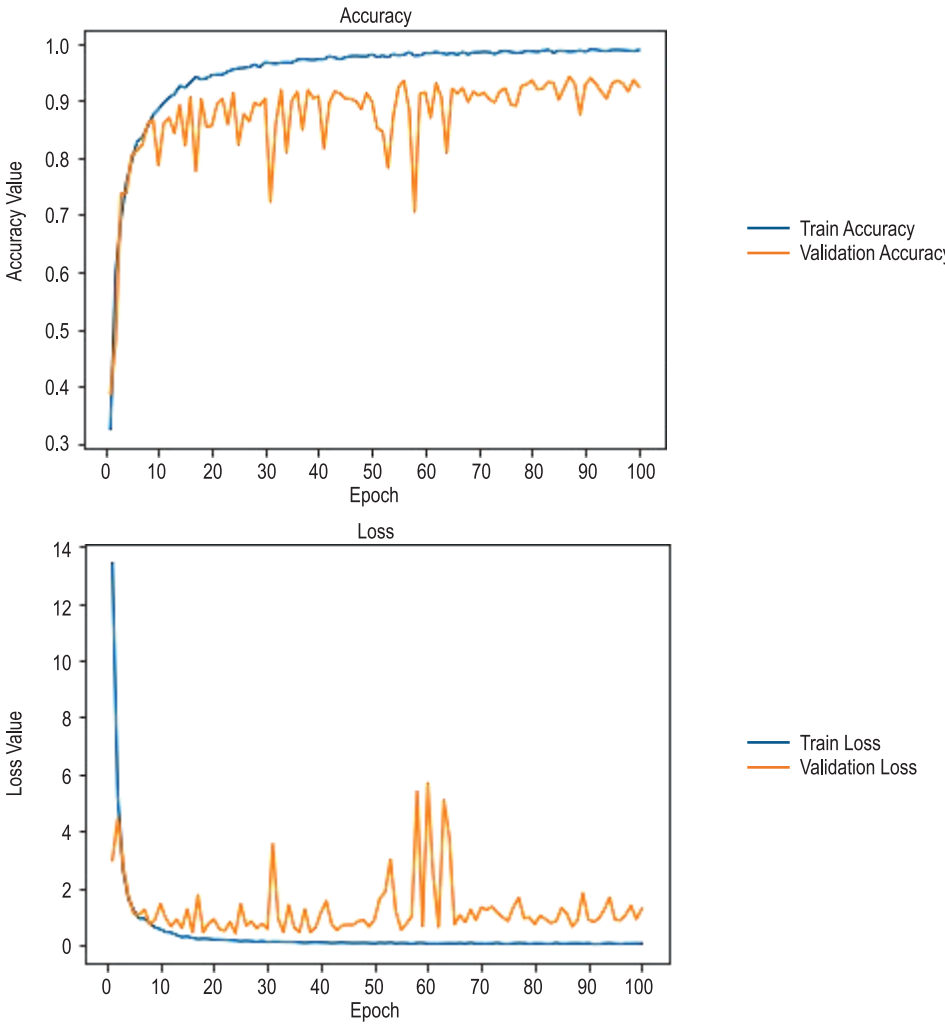

Fig. 8. The comparison of the learning process for the subset 1/2

Table 2

Classification for different subsets

Subset	Divide factor	Test subset accuracy [%]
1	1/2	93
2	1/4	83
3	1/8	76
4	1/16	71
5	1/32	56
6	1/64	43

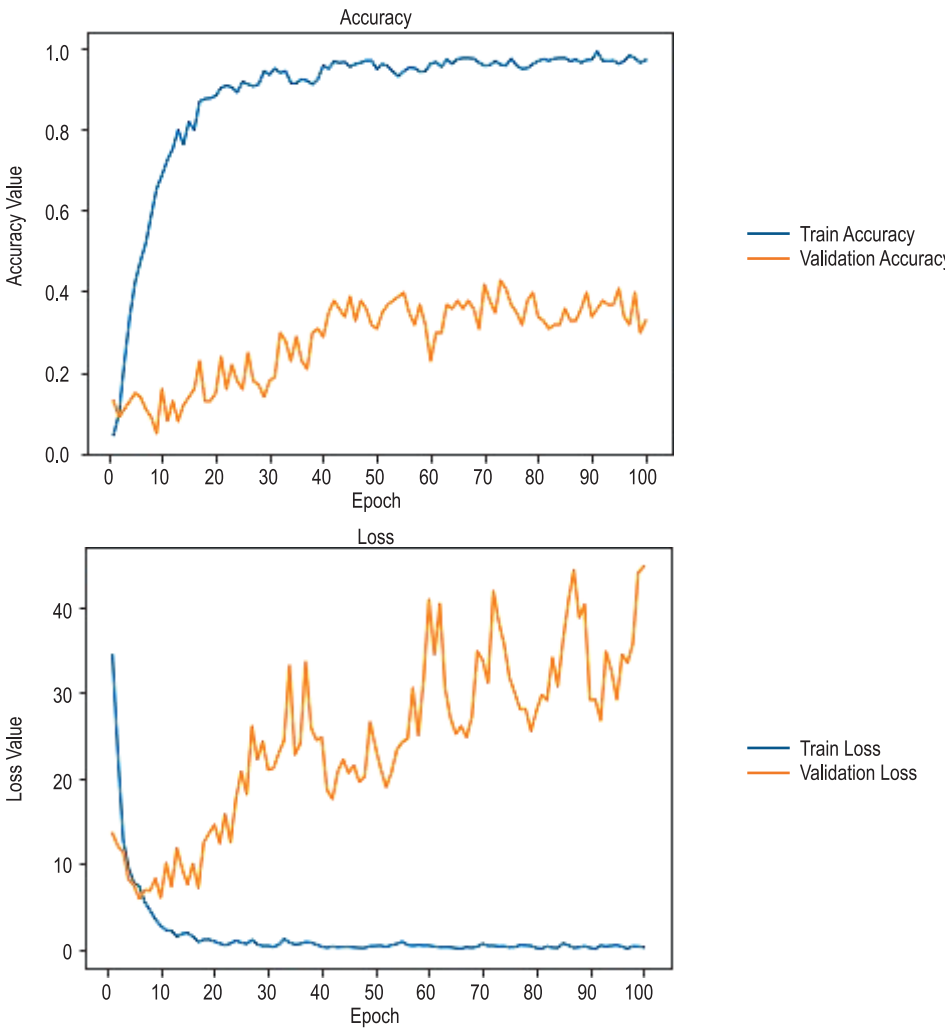

Fig. 9. The comparison of the learning process for the subset 1/64

Multiple binary convolution neural network model

One of the biggest problems with deep neural networks, apart from their high computational requirements, is their low flexibility. For a typical neural network, updating the input data sets or extending the data with new classes causes the need to re-train the model on the entire data set. For large data sets, such a process is time-consuming and requires considerable computational resources. The presented model is based on the observation that the network classifies well-known patterns, other data are not highly classified. By linking data into unique disjoint sets, with two pairs of different decision classes, we can create a data sub-model that will recognize a maximum of one class from the input set. Due to the fact that each of the sub-models contains one class in common with another sub-model, the recognition of data matching the pattern is unambiguous. If there are contradictions or ambiguities in the decisions of all appropriately selected pairs of sub-networks, it may be a signal that the test data does not belong to the domain of the learned data model, or the input data is contradictory or of poor quality. The proposed model allows an easy extension of the already existing model with new decision classes, and additionally, it does not require re-training the entire model. The model architecture is presented in Figure 10.

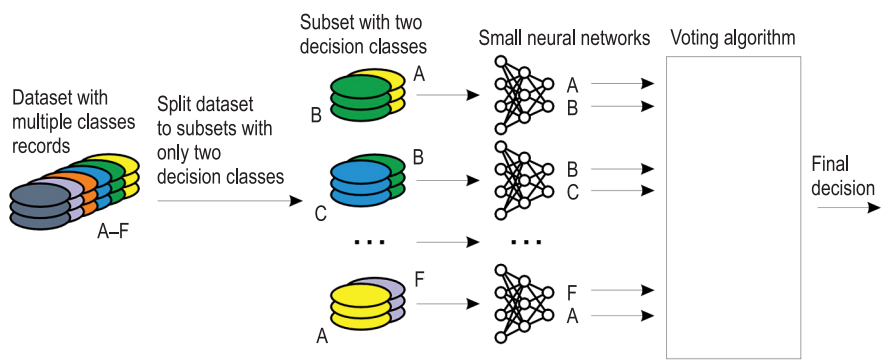

Fig. 10. Multiple binary convolution neural network architecture (MBCNN)

Since the subsets of the data are much smaller than the entire input set, neural subnets can also be much less complex than one large network for the entire set. Thanks to this, training a small subnet is much faster than training a large network for the entire data set. The process of updating the model, in the case of extending it with new decision classes, is presented in Figure 11. The input data for small neural networks belongs to two decision classes. Each of the subnets has two independent line outputs that can return values between 0 and 1, which determine the similarity of the tested input set to the two assigned classes. Due to the fact that the decision outputs are independent, the network can return

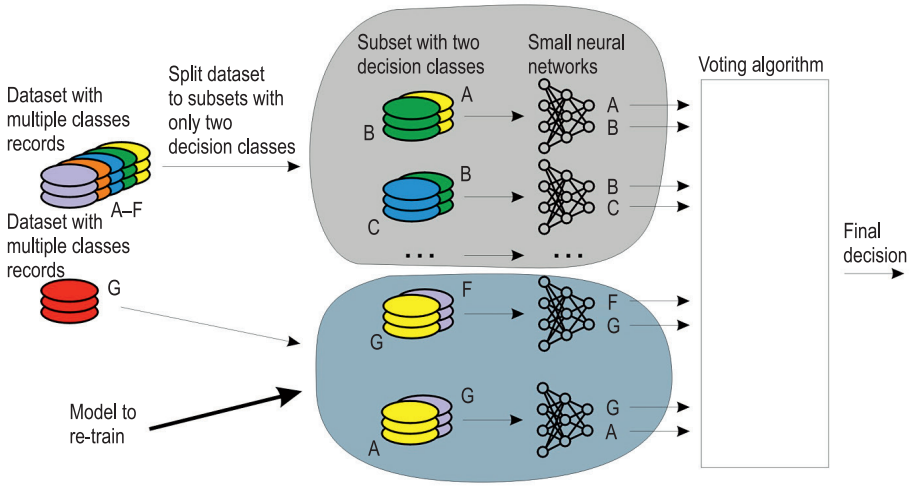

Fig. 11. Procedure of extending MBCNN with new decision classes

any values from the appropriate range on each of the outputs independently. Therefore, in order to be able to recognize which class the tested input sample belongs to, each output class is assigned to two disjoint binary subnets. Based on the analysis of network decisions, with the same labels on individual outputs, the decision algorithm ultimately decides what the final decision of the entire neural model is. The decision algorithm is based on the voting mechanism. The process of dividing the input file into binary input subsets is presented in Figure 12. There are only as many training sets as the original data output classes. Due to the fact that the subnets are independent of each other, the process of training them can be completely parallel. Thanks to this, when using multi-processor platforms, such models can be easily learned, without the need for expensive solutions based on GPU or ASIC units. Ordinary single large models are very difficult to scale into many smaller computing units.

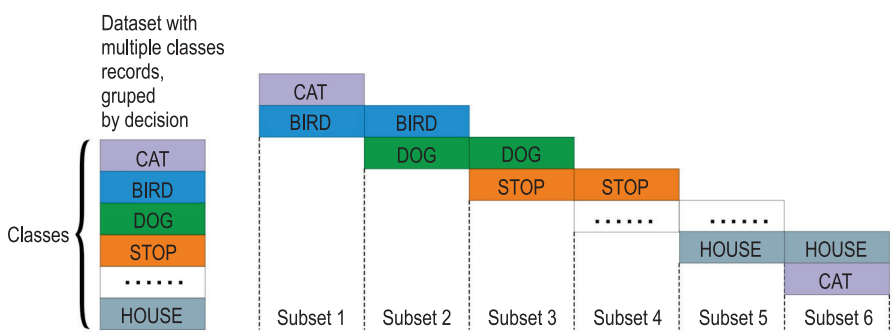

Fig. 12. The process of dividing the input dataset into binary input subsets

Voting algorithm

Generally, a voting classifier (CORNELIO et al. 2021, DONINI et al. 2018) is a machine learning model that uses a set of multiple models to predict an outcome based on the highest probability of a decision class of all component models. The mechanism consists in aggregating the results for each classifier, determining the decision based on the highest majority of votes. The mechanism is that, instead of creating separate dedicated models and finding the accuracy for each one, one model is created that trains against those models and predicts results based on their combined majority of votes for each output class. We can distinguish the hard or soft voting algorithm. In the case of hard voting, each of the component models returns one decision indicating a given class. The voting algorithm counts all decisions and approves the most numerous one. In the case of soft voting, each model returns the probability for each of the possible decisions, and then the voting algorithm to calculate the mean value for each class approves the class with the highest mean value. Examples of hard and soft voting algorithms are shown in Figures 13 and 14.

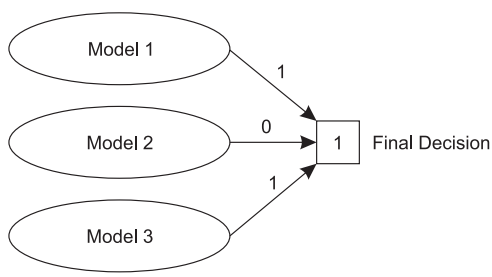

Fig. 13. Hard voting procedure

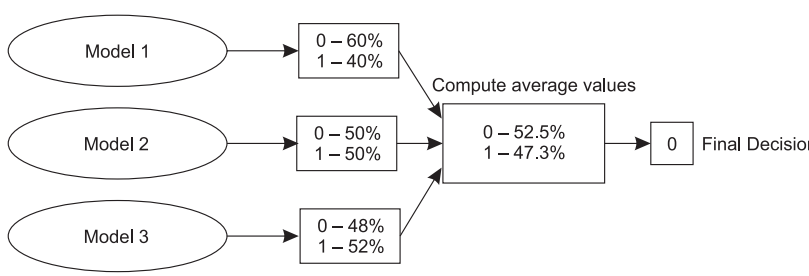

Fig. 14. Soft voting procedure

The voting algorithm presented in this paper is a modified version of the general algorithm. Many changes were introduced in the developed algorithm, the changes made are as follows:

– each of the component models is learned independently, in contrast to the typical voting algorithm, where the decision determined by voting is the basis for the adaptation of the weights of the entire model;

– each component models is trained for two data types, for each model there are different pairs, in a typical voting algorithm, the data is common, only the models are different;

– each component model has only two independent decision outputs that do not define the probability distribution of decisions (they do not sum up to 1.0)

In order to determine the final decision with modified voting algorithm, an analysis of the decisions of each of the neural sub-networks (models) should be performed. Each time we compare the responses of two sub-networks that have one of the outputs of the same class. As a result of the comparison, we can obtain one of three results: consistent, inconsistent, undefined. After analyzing all sub-network pairs for each class, we create a descending list of consistent results. Only sub-networks with consistent output are used, and the result with the greatest difference between the outputs of a given network is the winner. If it is not possible to determine the winner, the result of the classification is undefined. Possible scenarios of output values are presented in Figure 15.

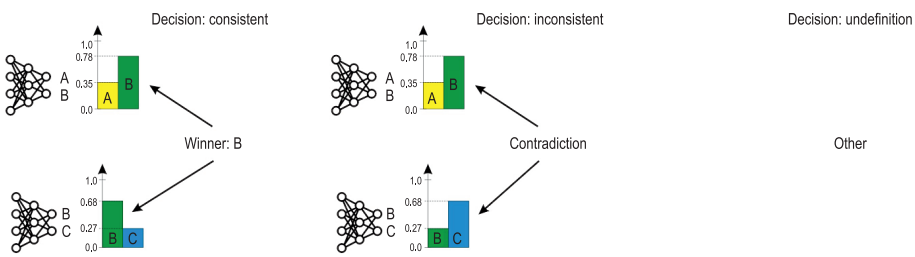

Fig. 15. Final decision voting procedure for MBCNN

The proposed algorithm determines the winners, it is defined as follows:

– we compare subnets where one of the output classes is common;

– we discard the output class if we obtained a classification other than consistent for a pair of subnets according to the following formula:

A, B – output classes,

a – minimum distance factor; β – minimum precision level,

$[A, B]$ = Sub-network,

if $[\text{abs}(A - B) < a]$ or $[\max(A, B) < \beta]$ then reject subnet

Value of α and β and beta should be between 0-1. In practice, the best results are obtained when α<0,5; 0,8> and β>0.8, this is depending of processed datasets:

– if the subnet set is empty, the network response is undefined, otherwise it goes to the next step;

– for each remaining pair of subnets, we calculate the accuracy coefficient for selected output class according to the following formula:

A, B, C – output classes, ε_0 – small positive value, A – main selected output class;

[A, B] = Sub-network 1,

[C, A] = Sub-network 2

$$\text{factor}(A) = \log\left(\frac{\max(A,B)}{\min(A,B)+\varepsilon_0}\right) + \log\left(\frac{\max(A,C)}{\min(A,C)+\varepsilon_0}\right)$$

– for the obtained coefficients factor (N), where N belongs to the set of all output classes, we return the coefficient with the highest value. The class name assigned to this factor determines the winning class

$$\text{winner class} = \arg(\max(\text{factor}(A), \text{factor}(B), \ldots)).$$

Results

The original dataset has data that can be assigned to one of 30 decision classes. Therefore, the input set was divided into 30 smaller pairs of data-sets, which in the next step were used to learn the appropriate subnets. Every input dataset was divided into the proportions of 80%, 10%, 10% for the training, validation and test sets. After training, the precision for the test set was calculated. The accuracy of the input dataset into pairs was presented in Table 3. Example results obtained for the sample "four/c948d727_nohash_2.wav" are presented in the Table 4. The final decision is determined by the class with the maximum sum of factor values, the values for the consistent classes are presented in the Table 5. To check the quality of the classification, 10% of the samples were randomly selected from the original database and the results were averaged. The obtained results for every class are presented in the Table 6.

Table 3

The division and accuracy of the input dataset into pairs

Id	Class name	Class name	Test accuracy [%]
1	2	3	4
1	four	two	96
2	two	wow	92
3	happy	wow	75

cont. Table 3

1	2	3	4
4	happy	tree	95
5	no	tree	92
6	bed	no	88
7	bed	off	95
8	bird	off	97
9	bird	nine	89
10	five	nine	90
11	eight	five	96
12	eight	zero	95
13	house	zero	97
14	go	house	94
15	go	stop	93
16	stop	up	89
17	down	up	95
18	down	seven	83
19	one	seven	95
20	one	right	93
21	dog	right	92
22	dog	left	93
23	left	on	93
24	on	six	96
25	six	three	95
26	cat	three	96
27	cat	sheila	96
28	sheila	yes	93
29	marvin	yes	95
30	four	marvin	95

Table 4

Results obtained for the sample "four / c948d727_nohash_2.wav"

Output pairs		Winner	Factor value	Consistent Pairs
1		2	3	4
bird	nine	bird	33.77759	–
cat	sheila	cat	20.59432	–
right	dog	dog	41.83123	1
dog	left	dog	24.77885	1
down	seven	down	22.89503	–
eight	zero	eight	63.37871	–
nine	five	five	73.93566	2

cont. Table 4

1		2	3	4
five	eight	five	22.52323	2
four	two	four	708.3964	3
four	marvin	four	708.3964	3
house	go	go	708.3964	4
go	stop	go	76.51535	4
zero	house	house	25.89251	–
left	on	left	24.43232	–
yes	marvin	marvin	71.25009	–
tree	no	no	708.3964	5
no	bed	no	55.25504	5
bed	off	off	26.95786	6
off	bird	off	28.59489	6
on	six	on	74.19561	–
seven	one	one	72.75239	7
one	right	one	59.21504	7
stop	up	stop	49.4408	–
six	three	three	3.200945	8
three	cat	three	2.961448	8
happy	tree	tree	18.8293	–
two	wow	two	15.06629	–
up	down	up	11.95415	–
wow	happy	wow	13.88821	–
sheila	yes	yes	30.20584	–

Table 5

The final decision is determined by the class with the maximum sum of factor values

Id	Class name	Factor
1	dog	66.61008
2	five	96.45889
3	**four**	**1,416.793**
4	go	784.9118
5	no	763.6515
6	off	55.55275
7	one	131.9674
8	three	6.162393

Table 6
The obtained results for the test data

Class name	Accuracy [%]
Four	65.5
Two	81.3
Wow	64.2
Happy	78.2
Tree	62.1
No	100
Bed	84
Off	85
Bird	86
Nine	65.2
Five	64.7
Eight	72.3
Zero	71.4
House	68.2
Go	56.3
Stop	67.8
Up	58.1
Down	82.3
Seven	85.7
One	83.2
Right	87.5
Dog	79.6
Left	95.2
On	96.4
Six	59.4
Three	98.2
Cat	55.2
Sheila	78.3
Yes	59.0
Marvin	97.5

Discussion

The presented approach has great potential in terms of using it to create a multi-class model, especially when we do not have access to the original data, and we want to extend the model with new classes. Compared to the large uniform model with full dataset, the presented architecture has a slightly lower average classification quality, some data from a given class are better classified and some worse than the uniform model. Weaker results for individual classes can be improved by selecting the appropriate parameters of the sub-model, as the entire network architecture consists of many independent subnets, which may have different architectures. However, if the results are compared to the full model, which contains a limited amount of input data, and obtains a similar quality of classification as the MBCNN architecture, we notice that such a unified model cannot have less than 1/8 of the original training set (12.5%). Meanwhile, in order to obtain the same quality of classification, single neural sub-networks in the presented architecture need from 5.29-7.35% (double the minimum and maximum size of a single class set in relation to all samples) of data from the input set. This is because a unified network needs to be able to access all samples during the training process, and reducing them to below a certain level drastically disrupts the ability to build a good model. In the case of the MBCNN architecture, each of the sub-networks only needs access for two appropriately selected classes, and the final decision is determined based on the analysis of the returned results from each sub-model, using a voting algorithm. Someone might notice that while individual sub-networks use a small amount of data for the whole set, there are as many data as decision classes, which means that all models are combined with a lot of input data. It is only worth paying attention to one quite significant difference in relation to the uniform neural model and MBCNN architecture. If we have input data that are not well balanced, i.e. we have large disproportions between the number of samples available for individual classes, e.g. the HAM10000 data set with dermatoscopic images, where the differences in the number of samples for individual classes reach 6,000%, then a uniform network will prefer decisions assigned to a larger class, which will result in a lot of false positive or false negative decisions for deficit classes, underestimating the average quality of classification for all samples. In the presented architecture, this problem is much easier to deal with. First, we do not need a dataset that is well balanced for all classes, in practice it is very difficult to achieve. It is enough for the sets in pairs to have a similar number of samples to be able to create sub-models correctly. By properly selecting the pairs, the impact of imbalance on the quality of the classification can be significantly reduced. Secondly, sub-networks in the MBCNN architecture are completely independent of each other, which means that the number of input samples for each sub-model is not that important, a certain dependency exists, because one decision class is always

associated with exactly two sub-networks. Third, we can obligatorily determine which pairs of decision classes will be assigned to specific sub-models, by trial and error we can obtain the best possible combination. In a uniform neural network, we have no influence on the distribution of the activities and sensitivity of individual neurons for individual classes, which also affects the quality of the decisions returned. We can extend the number of learning epochs, choose the way of initializing the weights to get the best results, but each time we have to re-learn on the full data set. In the case of the presented MBCNN architecture, we can optimize only the selected sub-network, not affect other sub-networks that correctly classify the data. The problem that may arise is classification errors if similar words are matched in pairs, e.g. "go", "no" for one sub-model. To improve the efficiency of the sub-models, the words should be matched in pairs so that they are not as close to each other as possible. On the other hand, the advantages, apart from extending with new classes without the need to re-learn the entire model, include the possibility to train the system only for selected already existing classes, without affecting the knowledge accumulated for other learned classes.

Conclusions

The article presents the architecture of a decision-making system composed of many smaller convolutional networks and a voting mechanism, whose task is to return the most appropriate decision or the lack of it, depending on the quality and type of input data. Based on the conducted experiments, it can be concluded that the presented architecture can also be as effective as a classic unified neural network, and in some tasks it can exceed it, e.g. when extending the model with new classes without the need to re-train the entire model, or the possibility of returning no decision. Another distinguishing feature of the presented architecture is the possibility of using completely different neural sub-networks or other classification methods to classify pairs of decision classes.

References

Cornelio C., Donini M., Loreggia A., Pini M.S., Rossi F. 2021. *Voting with random classifiers (VORACE): theoretical and experimental analysis.* Autonomous Agents and Multi-Agent Systems, 35(22). https://doi.org/10.1007/s10458-021-09504-y.

Donini M., Loreggia A., Pini M.S., Rossi F. 2018. *Voting with Random Neural Networks: a Democratic Ensemble Classifier.* RiCeRcA 2018. arXiv:1909.08996. https://doi.org/10.48550/arXiv.1909.08996.

Hoffmann J., Borgeaud S., Mensch A., Buchatskaya E., Cai T., Rutherford E., de Las Casas D., Hendricks L.A., Welbl J., Clark A., Hennigan T., Noland E., Millican K., van den Driessche G., Damoc B., Guy A., Osindero S., Simonyan K., Elsen E., Rae J.W.,

VINYALS O., SIFRE L. 2022.*Training Compute-Optimal Large Language Models*. https://arxiv.org/abs/2203.15556. https://doi.org/10.48550/arXiv.2203.15556.

O'SHEA K., NASH R. 2015. *An Introduction to Convolutional Neural Networks*. arXiv:1511.08458. https://doi.org/10.48550/arXiv.1511.08458.

SHAFAHI A., SAADATPANAH P., ZHU CH., GHIASI A. , STUDER C., JACOBS D., GOLDSTEIN T. 2020. *Adversarially Robust Transfer Learning*. ICLR 2020 Conference Blind Submission. https://openreview.net/pdf?id=ryebG04YvB.

WARDEN P. 2017. *Speech Commands: A public dataset for single-word speech recognition*. http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz.

WARDEN P. 2018. *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. arXiv:1804.03209. https://doi.org/10.48550/arXiv.1804.03209.

ZEGHIDOUR N., XU Q., LIPTCHINSKY V., USUNIER N., SYNNAEVE G., COLLOBERT R. 2019. *Fully Convolutional Speech Recognition*. arXiv:1812.06864. https://doi.org/10.48550/arXiv.1812.06864.