International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

Application of Deep Learning Methods for Recognizing and Classifying Culinary Dishes in Images

Iryna Tvoroshenko[1], Volodymyr Gorokhovatskyi[1], Oleg Kobylin[1], Arsenii Tvoroshenko[2]

[1]Department of Informatics
Kharkiv National University of Radio Electronics, Kharkiv, Ukraine
e-mail: iryna.tvoroshenko@nure.ua; volodymyr.horokhovatskyi@nure.ua; oleg.kobylin@nure.ua

[2]Hotelová škola Mariánské Lázně, p. o.
Mariánské Lázně, Česká republika

Abstract: *The paper deals with the actual tasks of computer vision – recognition and classification of objects in images. Deep learning methods based on neural networks are proposed as an alternative to existing methods. A new approach for effective recognition and classification of culinary dishes in images has been developed, which involves the use of the capabilities of the TensorFlow deep learning library and the features of the Convolutional Neural Network. A software application for recognizing and classifying culinary dishes has been developed. Testing of the proposed tool showed that 7 complete epochs of model training provided 77% accuracy, which is a fairly good result, given that one of the main problems in recognizing culinary dishes is interclass similarity. The prospect of further research is to create subclasses for existing general classes of dishes.*

Keywords—computer vision; classification; convolutional neural networks (CNN); deep learning; recognition

1. INTRODUCTION

Computer vision is a branch of computer science that studies the intelligent processing and analysis of visual data, such as images and videos [1-5]. Typical tasks of computer vision include image search, image recognition, image classification, visual materials restoration or generation, and others. One of the most interesting modern tasks that researchers are trying to solve is food recognition and classification. Today, most applications involving computer vision use neural network and deep learning technologies [6-9].

Various methods, technologies, and approaches to pattern recognition and classification have been developed [10-16]. Depending on the object to be recognized, the methods can provide different percentages of accuracy and efficiency on the same data [17-23]. In pattern recognition tasks, there is no single universal method that can be equally effective in all cases. Recognizing food and, for example, faces [18, 24] is the recognition of fundamentally different objects with different features and characteristics, which requires a special approach in both cases. Even for objects of the same class, different recognition and classification methods can be applied.

The development of proper nutrition methods has led to an increase in the number of mobile applications for calorie counting [4, 7, 8]. Most modern tools aimed at counting the calories of the food consumed by the user have a common workflow for this type of mobile application that uses a predefined database of dishes and products with average calorie content.

The first mobile apps for calorie counting were similar to a regular calculator, as the user had to write down the dish with all its characteristics: calories, calories, protein, fat, carbohydrates, and weight. Modern calorie counting apps are superior to their predecessors, but today's needs require the improvement of existing apps with artificial intelligence [25-28].

The object of research is the process of developing an application for recognizing and classifying culinary dishes.

The aim of this research is to develop a software application using deep learning methods for efficient recognition and classification of culinary dishes in images.

The research solved a number of problems:

1. Classical methods of image recognition and classification based on neural networks are analyzed.

2. The capabilities of the TensorFlow deep learning library are analyzed.

3. The features of the Convolutional Neural Network (CNN) are analyzed.

4. A new approach for efficient recognition and classification of culinary dishes in images has been developed.

5. A software application for recognizing and classifying culinary dishes was developed.

2. RELATED WORKS

Image pattern recognition methods work with different types of images and with different efficiency [1-8]. Considering the texture and the state of the image itself, an important problem that recognition methods can face is the presence of noise in the image [23-28]. The effect of noise on the operation of methods can lead to the fact that the method can extract more information features than necessary or actually available. The presence of noise in the image affects the formation of false keypoints and descriptors based on them.

To overcome this problem, methods such as BRISK spend extra time filtering out false descriptors. Other methods, such as ORB and AKAZE, may suffer from classification losses when some images are not found in their class or are mistakenly classified to the wrong class [19, 29].

To overcome the problems with finding information features in noisy images, special filters are most often used to filter out noise in the image and smooth it out. At the same time, the analysis of images that are too blurry is also ineffective for finding information features. With a high level of blurring, the computational cost for most methods increases significantly, and the number of keypoints decreases several times [15, 28]. In order to prevent this problem, the images to be recognized must be clear or have a small level of blur.

There are limitations on the size of the image to be recognized and classified [30, 31]. Large images must be of good quality, but are usually analyzed slowly; small images are inefficient for recognition due to the difficulty of identifying keypoints on them. Limitations on working with color and monochrome images are also important. Depending on the specifics and implementation, different recognition methods can work with either monochrome or color images [28-31].

Before choosing image recognition and classification methods, you need to understand what type of image you will have to work with because the methods have limitations on color, size, noise, brightness or dimness, background, and even the number of objects in the image, which somehow affects the process of finding information features [31]. Thus, there are many restrictions on the input data depending on the approach, method, and task. Depending on the data and method, the search for keypoints in the image is performed with different efficiency.

To recognize and classify fruit, which is a fairly large subclass of food, it has been proposed to use the KNN (K-Nearest Neighbors) method [8]. In their method, the researchers used a shape-based recognition strategy and primitive generation to estimate the parameters. First, adaptive smoothing was performed on the image to filter out Gaussian additive noise. Next, primitives are generated for estimation. The method also takes into account unripe fruit, which at the beginning of ripening can be very different in shape from a ripe sample. The recognition and classification accuracy for such examples was about 80%. The area and perimeter of the fruit are found for evaluation and further classification. Their values are calculated from the image pixels. The perimeter of the fruit is calculated by counting the boundary pixels, and the area is estimated by counting the total number of pixels inside the found boundaries. The method does not “see” the difference in the size of the photos, so the user is asked to make the objects approximately the same size. That is, if there is a photo of an apple somewhere far away on a tree and a photo of an apple nearby, then the photos need to be changed so that the objects are approximately the same size. Only then will the system calculate the area and perimeter and use these values for classification.

The Fruits Recognition system is based on the KNN method [9]. For the Fruit Recognition system, the method performs fruit classification using a distance measure; it selects the closest sample to the unknown fruit. The approach showed 90% accuracy in fruit recognition and classification [8]. However, the efficiency and accuracy of the method with more food groups will decrease significantly due to the large inter-class similarity and intra-class variations.

There are many other methods based on keypoint detection. Among them are BRIEF, ORB, BRISK, FREAK, AKAZE, LATCH, SIFT, SURF [19, 29]. The most popular is SURF.

SURF, or Speeded Up Robust Features, is a method widely used for image comparison, searching and classifying objects in a photo, and other tasks [28]. SURF is well suited for recognizing complex objects with a pronounced texture, such as culinary dishes. The effectiveness of the SURF method has been studied using fuzzy logic [12, 13]. In this case, fuzzy logic is used to determine the degree to which an object or a unique keypoint belongs to a reference class [14]. Using SURF as a method of object recognition can in some cases lead to significant time costs due to the large number of selected keypoints. For a mobile application, the system's processing time can become critical, but using a fuzzy logic approach can help avoid the problem of increasing processing time [15]. The SURF method using fuzzy logic has shown good results in face recognition [18, 24], but it is difficult to say whether it will show the same results in food recognition, especially on a very large sample.

SURF is one of the most effective computer vision methods, partially based on the Scale-Invariant Feature Transform (SIFT) method, and is used as a local detector of feature descriptors [32].

A descriptor is an identifier of a keypoint that distinguishes it from others. It is a set of data that describes an entity, object, or other unit, but is not the entity itself [30]. Usually, in an image, a descriptor describes information about a keypoint and its surroundings. A feature of SURF descriptors is the invariance property [19], which was also present in the SIFT method. It should be noted that small areas are less affected by deformation (image resizing, rotation, and other transformations). To be effective, the descriptor must have the property of invariance and keep the description of the keypoint the same under any size, rotation, or other change.

Some studies have proposed using the SURF method called Bag-of-SURF and the Spatial Pyramid Matching method [19]. With the Bag-of-SURF method, a code word dictionary is created, SURF feature vectors of images are extracted, and descriptors are built in 64-dimensional space, and grouped into 100, 200, or 300 clusters using the k-means method. The Spatial Pyramid Matching (SPM) method is used to segment the image into sub-regions and construct a histogram of local features. This approach proved to be effective, with 86% accuracy on a small dataset (6 categories, 53 food samples),

but on a large sample (111 categories, 2145 images), the accuracy of the method decreased several times [16].

An analysis of recent publications [4-8, 10, 13, 17, 33] has shown that methods using neural networks and machine or deep learning methods are best suited for recognizing and classifying culinary dishes [17]. There are many types of neural networks, some of which are actively used to solve pattern recognition problems. CNN networks require significant computational costs and powerful hardware (especially on large data samples), but for the task of recognizing and classifying culinary dishes, it is promising, as evidenced by the results of practical studies [34-37].

3. MATERIALS AND MODELS

Working with an image, and especially recognizing and classifying an object in it, is a rather difficult task for a computer. While for a human being it is an easy task to identify and instantly recognize what is depicted in a photo, a computer requires a number of complex methods, each of which can give different results on different data [27].

In food and culinary recognition, as mentioned earlier, the biggest problem is high cross-class similarity and a large number of representatives of the same class, which is why traditional approaches to recognition are considered ineffective.

To understand how the proposed method works, it is necessary to introduce the concepts of the Hessian and Hessian matrix.

A Hessian matrix is a square matrix defined by the formula below

$$H\big(f(x,y)\big) = \begin{vmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} \\ \frac{\delta^2 f}{\delta x \delta y} & \frac{\delta^2 f}{\delta y^2} \end{vmatrix}, \quad (1)$$

where H is the Hessian matrix; $f(x, y)$ is a function of the brightness gradient change [22].

The Hessian is a determinant of the Hessian matrix, used in the formation of the descriptor and works with the brightness of the gradient. The Hessian is determined by the formula below

$$det\ (H) = \frac{\partial^2 f}{\partial x^2} \cdot \frac{\partial^2 f}{\partial y^2} - \frac{(\partial^2 f)^2}{(\partial x \partial y)^2}. \quad (2)$$

It is with the help of the Hessian and Hessian matrix that SURF identifies keypoints. Special locations, such as corners, texture feature boundaries, etc., are identified through the Hessian, which reaches its extremes at locations with maximum changes in the brightness gradient.

To calculate brightness, an integral representation of an image is used. It is a matrix whose size corresponds to the size of the image. The elements of the matrix are calculated using a special formula

$$II(x,y) = \sum_{i=0,j=0}^{i\le x, j\le y} I(i,j), \quad (3)$$

where $I(i, j)$ is the brightness of a pixel.

In order to make the Hessian matrix invariant not only with respect to image rotation, but also with respect to scale in multiple dimensions, the SURF method uses filters of different scales. The formula for calculating the Hessian in SURF, taking into account filters of different scales, is given below

$$det\ (H_{approx}) = DD_{xx}DD_{yy} - (0.9 \cdot D_{xy})^2, \quad (4)$$

where DD_{xx}, DD_{yy}, D_{xy} are convolutions by filters; 0.9 is a coefficient that is theoretically justified and used to adjust the calculations.

The SURF method sets the threshold value of the Hessian. When searching for keypoints, the method goes through the pixels and looks for the maximum of the Hessian matrix determinant. If the pixel value is higher than the specified threshold, then the found pixel is considered as a candidate for a keypoint.

After the keypoint has been selected, the method generates descriptors based on it. A descriptor in SURF is a set of 64 numbers (sometimes 128) for each keypoint. Each number is the difference in the gradient around the keypoint (the keypoint itself has the maximum value). The invariance of the descriptor with respect to rotation is calculated as a random deviation of the gradient value from the average value within the radius of the keypoint and relative to the gradient direction. The area of descriptors calculation is determined by the size of the Hessian matrix.

Changes in illumination level also do not affect the search for keypoints due to its invariance to brightness shifts. The SURF method effectively finds both dark and light features in an image.

The steps for constructing a descriptor are shown in Fig. 1.

SURF is a very efficient and fast (unlike the SIFT method) method for finding key features in an image. It is well suited for finding objects in an image, but it does not work with an object by itself. The method analyzes the image as a whole and thus looks for features. SURF is good at recognizing complex objects with complex textures, but ineffective at recognizing simple objects with uniform, simple textures [9]. It is difficult for the method to find keypoints in such objects. However, the property of invariance in different cases, the efficiency of working with complex objects, and the speed of the method are important advantages of SURF as a recognition method. In recent years, among the many methods used for image recognition and classification, methods that use the capabilities of machine learning and neural networks have been very popular [11, 14].

A neural network is a mathematical model presented in a program form that was based on the principles of how real biological neurons in the brain work. The peculiarity of neural networks is the ability to “learn” different tasks by analyzing different examples. The use of neural networks for image

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

recognition is a very effective and popular method. The stages of creating and operating a neural network for recognition are shown in Fig. 2.

Kohonen neural networks are a set of neural networks based on the structure of the Kohonen layer [15]. The process of the Kohonen network is shown in Fig. 3.

In practice, it is quite common for data to be first processed by a Kohonen network and then passed to another, more efficient but computationally demanding neural network. For example, a convolutional neural network, this is most often used to solve complex problems of image recognition and classification.

Convolutional neural network or CNN refers to a type of multilayer feed forward networks in which the signal propagates in one direction. The principle of the network is inspired by the functioning of the visual cortex of animals and is aimed at efficient image recognition [21].

CNN is a part of deep learning technologies and is a part of many deep learning libraries, such as TensorFlow.

Deep learning is a subtype of machine learning and is based on learning data features and constructing a hierarchy of abstractions [22].

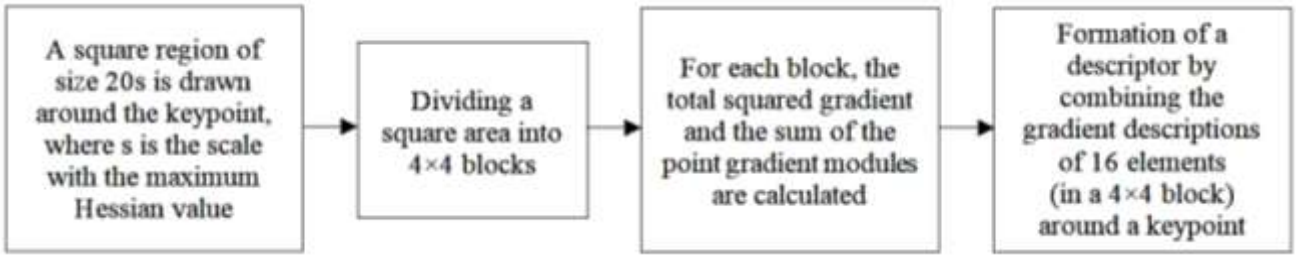

Fig. 1. *Steps in constructing a SURF descriptor*

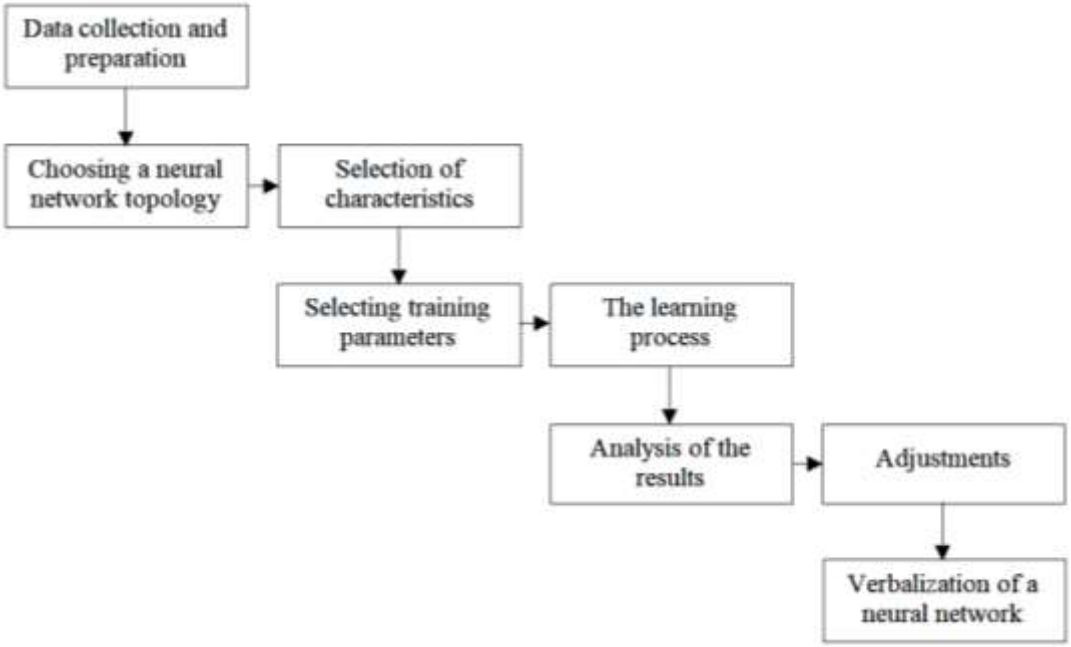

Fig. 2. *Stages of creating and operating a neural network*

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

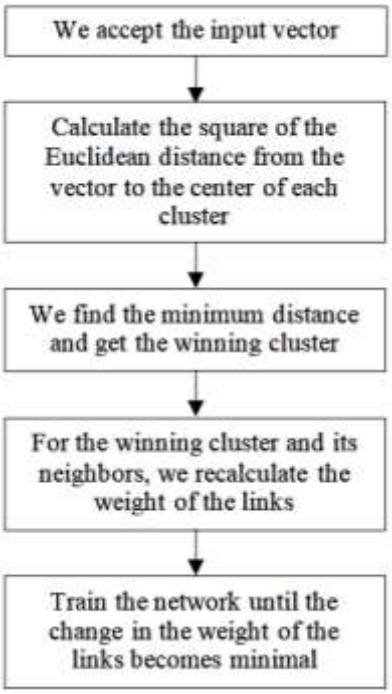

Fig. 3. *Stages of the Kohonen neural network operation*

While machine learning algorithms require structured data, deep learning networks rely entirely on their layers. Multilevel layers organize data into a hierarchy of concepts and learn from their own mistakes [26].

The idea of CNN architecture is to alternate convolutional and sub-sampling layers. The network gets its name from the convolution operation, in which each image fragment is multiplied by the convolution kernel element by element [25]. The result is added and written to a similar position of the original image.

CNN has a certain resistance to changes in scale, shift, rotation, angle, and other image transformations; the network has the property of invariance, which was inherent in the previously discussed SURF method. Three mechanisms help to ensure this property of the neural network: local feature extraction, formation of layers in the form of a set of feature maps, and the presence of a subsample [19].

Let's take a closer look at each of the mechanisms.

In local feature extraction, each neuron receives an input signal from the local receptive field on the previous layer. This is how the local feature is extracted. As soon as the network extracts a feature, its further exact location is no longer important because it is now determined relative to other found features.

The first layer in a convolutional neural network is the input layer. The network receives input data in the form of images. The input layer takes into account the two-dimensional structure of images and usually consists of three maps. Three because the image is represented in three main channels – red, blue, and green. If the image is represented in gray, only one map is created on the input layer.

The input pixel data is normalized from 0 to 1.

$$f(p, min, max) = \frac{p - min}{max - min}, \quad (5)$$

where f is the normalization function; p – pixel color in the range from 0 to 255; min – 0; max – 255.

Another important CNN mechanism is the formation of layers in the form of a set of feature maps. In the network topology, this feature is mainly adopted by the convolutional layer of the network. This layer is a set of such feature maps, each of which has a synaptic or scanning core. Simply put, it is a filter.

The number of maps is determined by the task statement and its requirements, but it should be noted that a large number of maps not only improves the quality and accuracy of recognition, but also increases the computational complexity [32].

Researchers advise to stick to a 1:2 ratio, in which the map of the previous layer is linked to two maps of the next one. The size of all maps of the convolutional layer is calculated using a special formula

$$(w, h) = (mW - kW + 1, mH - kH + 1), \quad (6)$$

where (w, h) is the size of the folding card; mW the width of the previous map; kW is the width of the kernel; mH the height of the previous map; kH is the height of the core.

As already mentioned, the synaptic or scanning kernel is a filter that goes through the map and finds the features of objects. The size of the kernel is set independently in the range from 3×3 to 7×7 but so that the size of the feature map is even [11]. This is necessary to prevent information loss when reducing the subsampling layer. The feature map on the convolutional layer is initially zero, and the weight of the kernels is randomly generated in the range from -0.5 to 0.5 [11]. A window of the size of the convolution kernel value passes through the image and at each iteration element by element multiplies the area under research by the kernel value. The sum of the multiplication result is written to the map of the next layer. This is how the convolution operation is performed, which can be represented as the following formula

$$(f * g)[m, n] = \sum_{k,l} f[m - k, n - l] * g[k, l], \quad (7)$$

where f is the initial image matrix; g – convolution kernel.

The result of convolution may differ depending on the method of processing the matrix borders. It can be smaller than the original image, the same size, or larger [34].

The third important CNN mechanism is subsampling. The subsampling layer comes immediately after the convolutional layer. It also has feature maps, but their number is the same as in the previous layer. The subsampling layer is needed to reduce the dimensionality of the feature maps. It filters out

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

unnecessary feature details and reduces the probability of overfitting. The kernel map of the subsampling layer has the dimensionality 2×2 and reduces the feature maps of the convolutional layer by half. This operation is called max pooling and uses the activation function. The activation function determines the dependence of the output signal on the input signal and displays numbers in the range (0,1) – less often (-1,1). The function shows the output value of the neuron and specifies whether the neuron should be active or can be ignored. That is, the activation function determines the functionality of the neural network and its training method.

In a convolutional neural network, the most commonly used activation function is ReLU – rectified linear unit or rectified linear function. Its derivative is either 0 or 1, it has no resource-intensive operations, no gradient growth or decay, or it facilitates fast learning, although it is not always reliable and is highly dependent on weight values.

The last type of CNN layer is a fully connected layer.

The purpose of this layer is to classify, by modeling a complex nonlinear function, as well as the output layer [33].

The general topology of CNN is shown in Fig. 4.

Thus, the use of CNNs is one of the best modern approaches to the task of pattern recognition and classification. Convolutional neural networks have the property of invariance, are efficiently trained, and are not very difficult to use, but they require a large amount of training data and computing resources. At the same time, there are a large number of open-source deep learning libraries that provide a wide range of tools. This makes it much easier to work with neural networks, especially convolutional ones. One of the most famous such libraries is the previously mentioned TensorFlow.

4. Deep Learning Methods of the TensorFlow Library for Pattern Recognition and Classification

TensorFlow is open-source machine learning and deep learning library designed specifically for solving high performance pattern recognition and classification tasks. Calculations in TensorFlow are performed using data-flow graphs, in which vertices are mathematical operations and edges are data represented by arrays or tensors.

In *n*-dimensional space, tensors are represented by an array of n^R of numbers. That is, a tensor is simply a mathematical object that does not depend on changes in coordinates, but its components depend and change according to certain mathematical laws. A flat vector and a matrix are special cases of a tensor in a certain space and with a certain rank. A tensor in a computation graph can be a number, a feature vector, an image, a set of object descriptions, or an array of images. In TensorFlow, computational graphs are executed in so-called sessions. A session is an object that is initialized as tf.Session and stores the necessary resources for graph execution [38]. These resources can include various helper classes, address space, etc. The library has two types of sessions: regular and interactive. The difference between them is only in the execution tool. An interactive session is more suitable for execution in the console and is immediately set as the default session.

An important step in TensorFlow is working with the data that will be used to train the neural network. The data for the neural model of image recognition and classification is a sample of images. As we have already mentioned, convolutional neural networks require large datasets for efficient training, so the larger the sample of images, the more efficiently the network will train, although it will take a lot of time and computing power to train on extremely large data.

TensorFlow makes it very easy to import or create your own dataset. A sample of images will be represented by an array of arrays. That is, an array of images, each of which is an array of pixels.

When working with images, the first challenge will be the problem of different sizes of different images. The neural network needs fixed data with the same image size as input. To overcome this problem, the TensorFlow library has a resizing method to create a special flattened layer of a one-dimensional array of images with the same size.

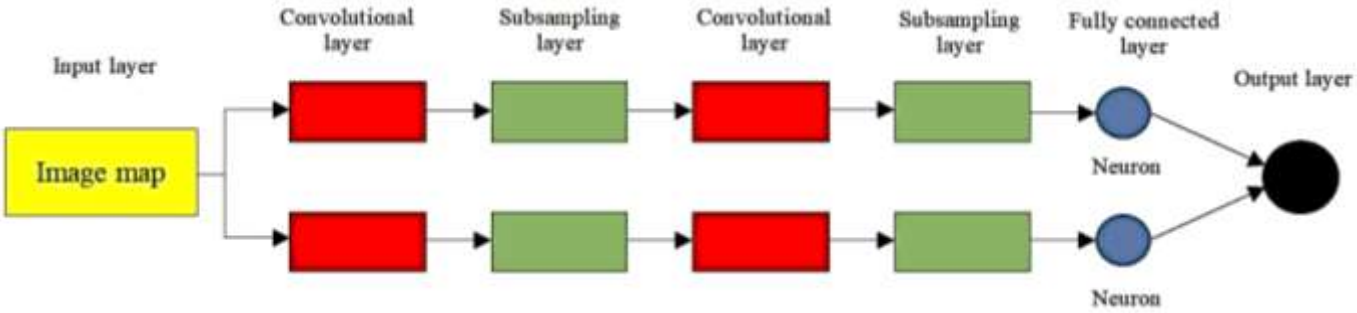

Fig. 4. *Topology of a convolutional neural network*

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

To do this, use the *input_shape* and *layers.Flatten* methods. The full entry looks like this: *tf.keras.layers.Flatten(input_shape(n,n,1)).*

TensorFlow makes it quite easy to work with color photos, although color is less important when it comes to classification. Therefore, at one stage of model development, you will need to convert the image colors to grayscale.

As with scaling, TensorFlow can use a special sub-library Scikit-Image, in which the *color* module and the *rgb2gray()* function play an important role. This function works with the color map, which has already been discussed in the previous paragraph.

Finally, after pre-processing the images from the training set, the neural network is modeled. The computation graph is created using the *Graph()* function. The created graph itself does not calculate anything, since the creation function does not accept any variables. *Graph()* defines the operations that will be performed further. You can add the necessary operations to the graph. You need to create a neural network model, define the loss function, optimizer, and metric at compile time. With the help of TensorFlow methods, this is done quite quickly.

First, you need to set placeholders for input data and labels. Placeholders are uninitialized variables that will be initialized by the created session when it is launched using the *session.run()* method. In this case, placeholders receive the values of the dataset. That is, they parameterize the computation graph and mark the places for substituting external values. Using a simple example [38], a list of placeholders and sessions is shown in Fig. 5 and a graph of computations based on them in Fig. 6.

```
x = tf.placeholder(tf.float32)
f =  1 + 2 * x + tf.pow(x, 2)
sess.run(f, feed_dict={x: 10})
>>> 121.0
```

Fig. 5. *Listing of placeholder and session usage*

The x and *y* variables are simply additional parameters that can be replaced by edges. First, a placeholder is created, and the graph of the expression in the variable *f is* created on its basis. Tensors of any size can be fed as input, since the size is not explicitly specified, but it is important to specify the type of tensor. The *feed_fict()* function is a dictionary of session parameters. The session itself is started through *sess.run().*

When placeholders are created and a session is defined, you need to smooth the input data using the *flatten()* function. After that, a layer that generates logits is created.

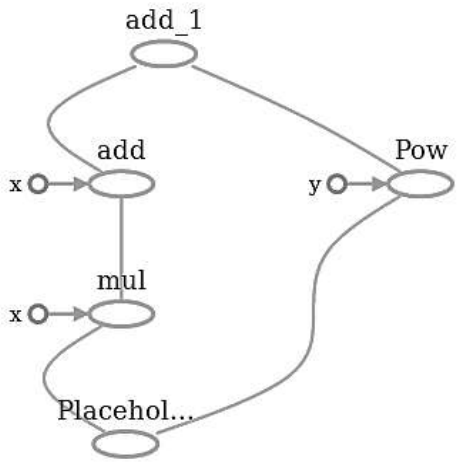

Fig. 6. *Computation graph for operations created by a placeholder and a session*

Logits are functions that work with the output of the neural network layers and use a relative scale to check the linearity of the unit of measurement [38].

After modeling the layers of the convolutional neural network, the loss function is determined. The loss function is set by the developer independently, depending on the needs of the problem.

It is a key component of the learning process, as it measures the difference between the model's output value and the target value, i.e. it is an important indicator of model performance. The smaller the function, the stronger the model [37]. During the training process, the loss function tells TensorFlow whether the forecast result is better or worse than the target result [38].

TensorFlow has several types of built-in loss functions: L1 regular loss function, L2 regular loss function or Euler loss function, sigmoidal cross-entropy loss function, Softmax cross-entropy loss function, and many others.

For the task of classifying images of culinary dishes, various studies have recognized the last function as the most effective – the Softmax cross-entropy loss function. A listing of the function code from the TensorFlow library is shown in Fig. 7 [38].

```
loss = tf.losses.sparse_softmax_cross_entropy(
    labels=train_y, logits=train_logits)
tf.summary.scalar('loss', loss)
```

Fig. 7. *Code for using the Softmax cross-entropy loss function*

This function is applied to the non-normalized output and calculates the losses for one target classification.

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

The output becomes a probability distribution using the *softmax_cross_entropy_with_logits()* function. This helps to facilitate the entry into cross-entropy for further calculations. The cross-entropy formula is as follows:

$$\sigma(Z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, \ldots, K. \quad (8)$$

Based on cross-entropy, the implementation of the Softmax cross-entropy loss function is implemented internally as follows:

$$L_i = -f_{y_i} + log \sum_j e^{f_j}. \quad (9)$$

After selecting the loss function, you need to install the learning optimizer. As well as loss functions, TensorFlow implements many methods of training optimization. Among them: Stochastic Gradient Descent (SGD), ADAM, RMSprop, and many others [38].

An optimizer is a method that helps speed up and improves the learning process. If the loss function is a measure of how well the model predicts the expected outcome, then to minimize the loss function, there must be a function that updates the parameters (or in other words, the weights of the network links) to increase accuracy [9].

Such a function is an optimizer that changes the values of parameters (weights of connections) and learning rates. This increases the correctness, accuracy, and speed of the neural network during training.

TensorFlow most often uses Stochastic Gradient Descent, which has a lot of fluctuations during training, which is well optimized in combination with the Momentum impulse optimizer.

A list of SGD usage in combination with a pulse optimizer is shown in Fig. 8.

The Momentum optimizer is implemented as follows:

$$v_{dW} = 0{,}9v_{dW} + (1 - 0{,}9)dW, \quad (10)$$

$$v_{db} = 0{,}9v_{db} + (1 - 0{,}9)db, \quad (11)$$

$$W = W - \alpha v_{dW}, b = b - \alpha v_{dW}. \quad (12)$$

Using the reduce_mean() method, after initializing the selected optimizer, we set the accuracy metrics. The *reduce_mean()* method performs convolutions of the optimization function, calculates the average value of the elements over the course of tensor changes. At this stage, the creation of a neural network in TensorFlow is completed.

The created neural network is launched in a session, starts to go through training cycles, generates results, and is evaluated. The evaluation is performed by selecting random images and comparing the predicted result with the expected one [38]. As you can see, the TensorFlow deep learning library provides convenient and efficient methods for creating neural networks and performing image recognition and classification.

5. Software Simulation Results

The research used the classic TensorFlow library for the Python programming language and the TensorFlow.js library for working with the JavaScript programming language. To quickly form a neural network, we used the interface of the Keras library, which is an add-on to TensorFlow. Keras works with network layers, functions, and optimizers. To accelerate the learning process, the neural network uses the weights of the open model Inceptionv3, which is designed to recognize various models. The weights are used in the convolutional network developed with TensorFlow and Keras and are customized for the selected data.

To develop a mobile and web application, you need to design the server and client parts of the application. To accomplish this task, we used the JavaScript programming language and related frameworks. The server side of the application stores the created neural network model, which is converted from Keras to TensorFlow.js format using TensorFlow.js. This is necessary for the correct processing of the model and its transfer between the server and the client, which will be written in JavaScript. The server itself is created using the following technologies: Express.js, TypeScript, Node.js, and Webpack. TypeScript, Webpack, and the React framework were chosen to develop the client side. Thus, all of the above tools were used to create a mobile and web application for recognizing and classifying culinary dishes in images.

To develop the application, we used the Food101 dataset with 101 classes of dishes as the source data. In total, the set contains 101000 images of culinary dishes. Each class has 750 training samples and 250 test samples. The images contain noise, and their maximum size does not exceed 512 pixels. A fragment of images from the training set is shown in Fig. 9.

When you start working with a training set, you need to upload it to the project and unzip it. In this development, unpacking took some time because the size of the training set is almost 5 gigabytes.

```
optimizer = tf.train.MomentumOptimizer(learning_rate=lr, momentum=MOMENTUM)
training_op = slim.learning.create_train_op(
    loss, optimizer, global_step=global_step)
```

Fig. 8. *Listing of the Momentum optimizer in TensorFlow*

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

Fig. 9. *Fragment of images from the training set*

Standard methods of the Keras library were used for unpacking. After unpacking all the files, it is necessary to divide the images into training and test images using the *prepare_data* function, the code of which is shown in Fig. 10.

```
def prepare_data(filepath, src,dest):
  classes_images = defaultdict(list)
  with open(filepath, 'r') as txt:
      paths = [read.strip() for read in txt.readlines()]
      for p in paths:
        food = p.split('/')
        classes_images[food[0]].append(food[1] + '.jpg')

  for food in classes_images.keys():
    print("\nCopying images into ",food)
    if not os.path.exists(os.path.join(dest,food)):
      os.makedirs(os.path.join(dest,food))
    for i in classes_images[food]:
      copy(os.path.join(src,food,i), os.path.join(dest,food,i))
```

Fig. 10. *Function for dividing images into training and test images*

At this stage, this is all that is required for preliminary processing of the sample. In the future, you need to perform a few more operations on the images, for example, resizing the images to a single constant value for further work with the neural network.

To set up the weights of the pre-trained Inceptionv3 model, the Food101 sample was used. The use of a pre-trained model with predefined weights allows you to speed up the training process many times over and save not only time but also computing resources.

You only need to adjust the model weights for the sample and add a few new layers to continue working, no need to create everything from scratch. The new *train_model* function will help you start the session. Resize all images to 300×300. For the training and test images, we generate three classes of different image variations from:

1. Zoom.
2. By turning.
3. By shear.

Initialize the Inceptionv3 model. Set the activation function ReLu for it. The Softmax cross-tropic loss function is used as a loss function. The stochastic gradient descent or SGD is used as the neural network optimizer.

The general process of setting up a neural network model is shown in Fig. 11 and Fig. 12.

After setting up the model, we set the number of classes. In our research, we used 101 classes. Enter the number of epochs (iterations) of training, for example, 20 epochs. The code listing for starting the training and finalization is shown in Fig. 13.

The resource-intensive model training process begins. Each time, the model is fed with data from the training set, and the training accuracy is analyzed on the test data. With each iteration, the model's recognition and classification accuracy increases. The training results are shown in Fig. 14.

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

```
def train_model(n_classes,num_epochs, nb_train_samples,nb_validation_samples):
  K.create_session()

  img_width, img_height = 300, 300
  train_data_dir = 'food-101/train_mini'
  validation_data_dir = 'food-101/test_mini'
  batch_size = 16
  bestmodel_path = 'bestmodel_'+str(n_classes)+'class.hdf5'
  trainedmodel_path = 'trainedmodel_'+str(n_classes)+'class.hdf5'
  history_path = 'history_'+str(n_classes)+'.log'

  train_datagen = ImageDataGenerator(
      preprocessing_function=preprocess_input,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True)

  test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

  train_generator = train_datagen.flow_from_directory(
      train_data_dir,
      target_size=(img_height, img_width),
      batch_size=batch_size,
      class_mode='categorical')

  validation_generator = test_datagen.flow_from_directory(
      validation_data_dir,
      target_size=(img_height, img_width),
      batch_size=batch_size,
      class_mode='categorical')
```

Fig. 11. *Listing of code to prepare model training*

```
  inception = InceptionV3(weights='imagenet', include_top=False)
  x = inception.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(128,activation='relu')(x)
  x = Dropout(0.2)(x)

  predictions = Dense(n_classes,kernel_regularizer=regularizers.l2(0.005), loss='softmax')(x)

  model = Model(inputs=inception.input, outputs=predictions)
  model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accu
racy'])
  checkpoint = ModelCheckpoint(filepath=bestmodel_path, verbose=1, save_best_only=True)
  csv_logger = CSVLogger(history_path)

  history = model.fit_generator(train_generator,
                    steps_per_epoch = nb_train_samples // batch_size,
                    validation_data=validation_generator,
                    validation_steps=nb_validation_samples // batch_size,
                    epochs=num_epochs,
                    verbose=1,
                    callbacks=[csv_logger, checkpoint])

  model.save(trainedmodel_path)
  class_map = train_generator.class_indices
  return history, class_map
```

Fig. 12. *Listing of the code for configuring the Inceptionv3 model*

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

```
n_classes = 101
epochs = 20
nb_train_samples = train_files
nb_validation_samples = test_files

history, class_map = train_model(n_classes,epochs, nb_train_samples,nb_validation_samples)
print(class_map)
```

Fig. 13. *Listing of code for setting epochs and starting training*

```
Epoch 1/20
4734/4734 [==============================] - 17810s 4s/step - loss: 5.0469 - accuracy: 0.0412 - val_loss: 3.7701 - val_accuracy: 0.3359

Epoch 00001: val_loss improved from inf to 3.77012, saving model to bestmodel_101class.hdf5
Epoch 2/20
4734/4734 [==============================] - 17791s 4s/step - loss: 3.8104 - accuracy: 0.2786 - val_loss: 2.4422 - val_accuracy: 0.5539

Epoch 00002: val_loss improved from 3.77012 to 2.44223, saving model to bestmodel_101class.hdf5
Epoch 3/20
4734/4734 [==============================] - 17808s 4s/step - loss: 2.9014 - accuracy: 0.4360 - val_loss: 1.9032 - val_accuracy: 0.6463

Epoch 00003: val_loss improved from 2.44223 to 1.90316, saving model to bestmodel_101class.hdf5
Epoch 4/20
4734/4734 [==============================] - 17818s 4s/step - loss: 2.4352 - accuracy: 0.5235 - val_loss: 1.6446 - val_accuracy: 0.6968

Epoch 00004: val_loss improved from 1.90316 to 1.64460, saving model to bestmodel_101class.hdf5
Epoch 5/20
4734/4734 [==============================] - 17821s 4s/step - loss: 2.1539 - accuracy: 0.5820 - val_loss: 1.4584 - val_accuracy: 0.7343

Epoch 00005: val_loss improved from 1.64460 to 1.45843, saving model to bestmodel_101class.hdf5
Epoch 6/20
4734/4734 [==============================] - 17813s 4s/step - loss: 1.9443 - accuracy: 0.6229 - val_loss: 1.3502 - val_accuracy: 0.7540

Epoch 00006: val_loss improved from 1.45843 to 1.35024, saving model to bestmodel_101class.hdf5
Epoch 7/20
4734/4734 [==============================] - 17812s 4s/step - loss: 1.7936 - accuracy: 0.6541 - val_loss: 1.2665 - val_accuracy: 0.7685

Epoch 00007: val_loss improved from 1.35024 to 1.26646, saving model to bestmodel_101class.hdf5
Epoch 8/20
4153/4734 [=========================>....] - ETA: 34:53 - loss: 1.6643 - accuracy: 0.6792
```

Fig. 14. *Model training results*

As we can see in Fig. 14, the model has passed incomplete 8 epochs out of the set 20, and the model training stopped due to the lack of computing resources. The training process for incomplete 8 iterations took almost 5 hours. In addition, the 7 complete epochs showed 77% accuracy of the model, which is a fairly good result, but indicates a rather large percentage of recognition and classification errors. This is due to the fact that one of the main problems in recognizing culinary dishes is cross-class similarity.

The trained model stored on the server is transferred to the client side using TensorFlow.js. After the model has been successfully loaded, the client side is loaded with the culinary labels.

The client side involves several stages:

1. Downloading models and labels from the server.
2. Obtaining an image for further analysis.
3. Analysis and output of the result.

You can get an input image either by downloading a graphic file from the system or through a media stream (camera). During the research, we implemented a component that receives a media stream and outputs it using the video tag, and added two methods for obtaining an image:

1. Save a slice of the media stream as an image.
2. Upload an image from the file system.

The image is saved in the app.

Analyzing the saved image provides:

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

1. Scaling the image to 300×300.

2. Mapping data from the range of 0 to 255 to the range of -1 to 1.

3. Change the tensor rank to 0.

After analyzing the saved image, the data is transferred to the model, and it returns an array of numbers (the percentage of the image belonging to the corresponding class). At this stage, the work with the client side is finished.

The developed application has the form shown in Fig. 15.

Fig. 15. *Screen form of the finished application*

Depending on the features of the image, the app produces different recognition results for the same object. For example, depending on the color, image quality, or contrast, the app can identify a "steak" as a "chocolate cake" and vice versa. Regarding the number of dishes that can be recognized by the application at one time in one image, it should be noted that the application is designed to recognize and classify one dish. However, in practice, if there are several dishes in the image, the app will display all or almost all of the dishes in the image in the top 10 of the resulting list. Only one dish will be the main focus and the highest percentage of probability (Fig. 16).

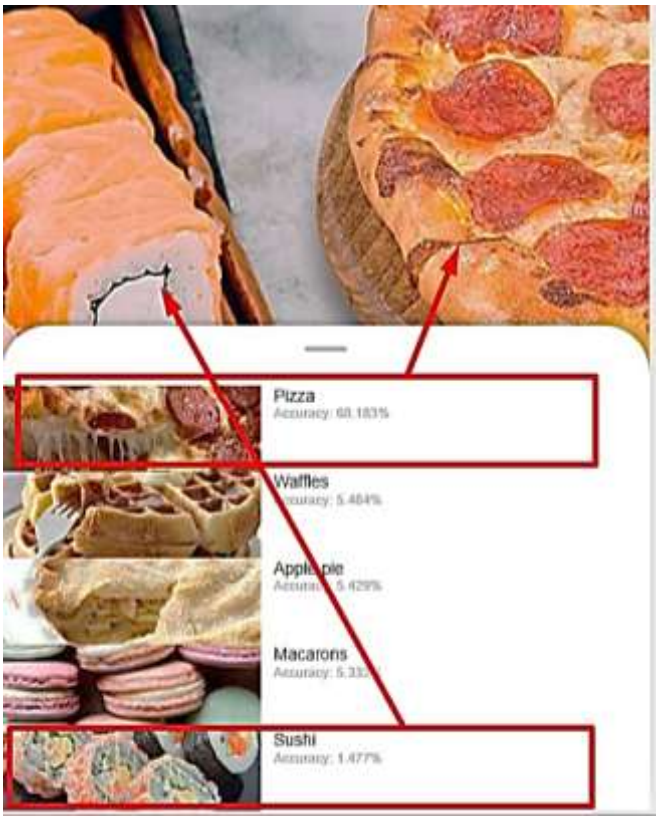

Fig. 16. *The result of recognition and classification of an image with several dishes*

As you can see in Fig. 16, the processed image contains 2 dishes – "sushi" and "pizza". The image analysis resulted in a list of dishes. Both dishes are in the top 5, but the "pizza" dish received more focus and percentage of recognition. This result is explained by the fact that the model compares the input image with all the classes it knows and gives the percentage of similarity for each class (101 classes in total). If we sum all the similarity values, we get 100%.

In addition, if the model does not know the culinary dish detected in the image, the system will distribute the percentage of similarity among all classes, but the values for each class will be very small.

6. Discussion

The most important criterion for model and application quality is the accuracy of input image recognition. At the training stage, the model passed incomplete 8 epochs out of 20, and its accuracy is currently 77%. After the training stage, we analyzed the reduction of losses and the increase in accuracy during training and validation of the model. The results of the analysis are shown in Fig. 17 and Fig. 18.

In Fig. 17, the blue color indicates the result of training, and the green color indicates the result of validation.

International Journal of Academic and Applied Research (IJAAR)
ISSN: 2643-9603
Vol. 7 Issue 9, September - 2023, Pages: 57-70

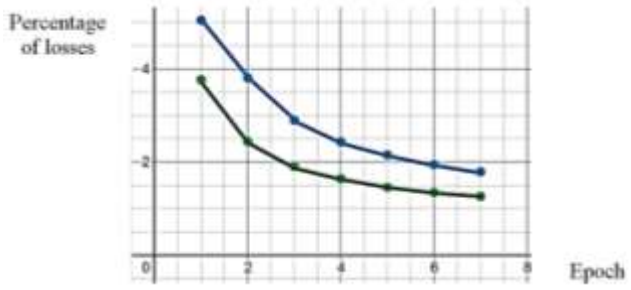

Fig. 17. *Graph of loss reduction during model training and validation*

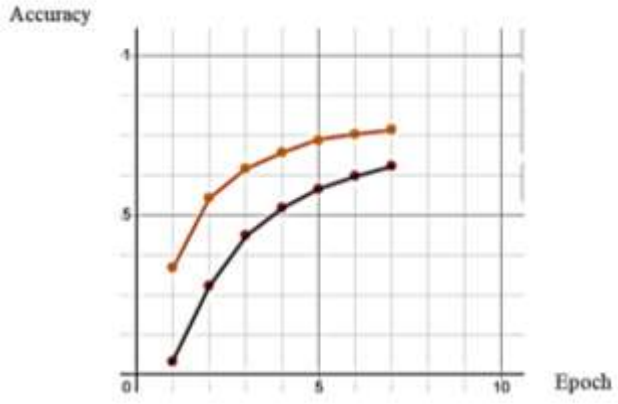

Fig. 18. *Graph of model accuracy increase during training and validation*

In Fig. 18, the red color indicates the result during training, and the orange color indicates the result during validation. As can be seen in Fig. 17 and Fig. 18, with each epoch, the percentage of model losses decreased and the percentage of accuracy increased. In addition, the results improved depending on the stage: already from the first epoch, the difference between the training and validation results is noticeable. After each iteration of training, validation at the same step yielded a much better result.

7. Conclusion

A new approach to searching for objects in an image using deep learning methods based on convolutional neural networks is proposed, which has shown high data processing speed and sufficient efficiency. An experiment for the task of recognizing several objects in an image gave the percentage of similarity for each class. The effectiveness of the developed approach can be enhanced by improving training and choosing parameters such as the number of network layers, setting functions and weights, and optimizers.

The novelty of the research lies in the development and experimental testing of deep learning methods by improving the tools of convolutional neural networks, which helps to increase the speed of recognition and classification and ensures sufficient efficiency.

The practical significance of the work is to increase the depth of visual data analysis, as well as the speed of recognition and classification, to confirm the effectiveness of the proposed approach on specific examples, to create an application software tool for studying and implementing recognition and classification methods in the latest computer vision systems.

The prospect of further research to improve the application is to create subclasses for existing general classes of dishes.

8. Acknowledgment

The authors acknowledge the support of the Department of Informatics, Kharkiv National University of Radio Electronics, Ukraine, in numerous help and support to complete this paper. The authors are grateful to Karyna Temchur for his participation in the implementation of software modeling.

9. References

[1] Myers, A. et al. (2015). Im2Calories: towards an automated mobile vision food diary, in Proc. IEEE Int. Conf. Comput. Vis., Santiago, Chile, pp. 1233-1241.

[2] Tvoroshenko, I., & Zarivchatskyi, R. (2020). Analysis of existing methods for searching object in the video stream, in Proc. VI Int. Sci. Practic. Conf. «About the problems of science and practice, tasks and ways to solve them», Milan, Italy, pp. 500-505.

[3] Cannell, R. C. et al. (2002). Online evaluation of a commercial video image analysis system (Computer Vision System) to predict beef carcass red meat yield and for augmenting the assignment of USDA yield grades. United States Department of Agriculture, J. Animal Sci., vol. 80, no. 5, pp. 1195-1201.

[4] Sahoo, D. et al. (2019). FoodAI: Food image recognition via deep learning for smart food logging, in Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov & Data Min., pp. 2260-2268.

[5] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Salt Lake City, UT, USA, pp. 7132-7141.

[6] Pant, G., Yadav, D. P., & Gaur, A. (2020). ResNeXt convolution neural network topology-based deep learning model for identification and classification of Pediastrum, Algal Res., vol. 48, p. 101932.

[7] Sun, J., Radecka, K., & Zilic, Z. (2019). FoodTracker: a real-time food detection mobile application by deep convolutional neural networks. [Online]. Available: https://arxiv.org/pdf/1909.05994.pdf.

[8] Aguilar, E., Bolaños, M., & Radeva, P. (2017). Food recognition using fusion of classifiers based on CNNs, in Int. Conf. Image Anal. Proc., Springer, Cham, pp. 213-224.

[9] Bezdan, T., & Bacanin, N. (2019). Convolutional neural network layers and architectures, in Sinteza 2019 – Int. Sci. Conf. Inf. Technol. Data Relat. Res., Belgrade, Singidunum University, Serbia, pp. 445-451.

[10] Seng, W. C., & Mirisaee, S. H. (2009). A new method for fruits recognition system, in 2009 Int. Conf. Electr. Eng. Inform., Bangi, Malaysia, pp. 130-134.

[11] Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: An overview and

application in radiology, Insights Imaging, vol. 9, pp. 611-629.

[12] Kucherenko, Y. I., Filatov, V. A., Tvoroshenko, I. S., & Baidan, R. N. (2005). Intellectual technologies in decision-making technological complexes based on fuzzy interval logic, East Eur. J. Adv. Technol., vol. 2, pp. 92-96.

[13] Konaje, N. K. (2016). Food recognition and calorie extraction using bag-of-surf and spatial pyramid matching methods. [Online]. Available: https://cvgl.stanford.edu/teaching/cs231a_winter1415/prev/projects/food.pdf.

[14] Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst., pp. 1097-1105.

[15] Gorokhovatskyi, V., & Tvoroshenko, I. (2020). Image Classification Based on the Kohonen Network and the Data Space Modification, in CEUR Workshop Proc.: Comput. Modeling Intell. Syst. (CMIS-2020), vol. 2608, pp. 1013-1026.

[16] Kagaya, H., Aizawa, K., & Ogawa, M. (2014). Food detection and recognition using convolutional neural network, in Proc. 22nd ACM Int. Conf. Multimed., pp. 1085-1088.

[17] Ciocca, G., Napoletano, P., & Schettini, R. (2016). Food recognition: a new dataset, experiments, and results, IEEE J. Biomed. Health Inform., vol. 21, no. 3, pp. 588-598.

[18] Nour, N., Elhebir, M. & Serestina V. (2020). Face expression recognition using convolution neural network (CNN) models, Int. J. Grid Comput. & Appl. (IJGCA), vol. 11, no. 1/2/3/4. [Online]. Available: https://aircconline.com/ijgca/V11N4/11420ijgca01.pdf.

[19] Tareen, S. A. K., & Saleem, Z. (2018). A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK, in 2018 Int. Conf. Comput., Math. Eng. Technol. (iCoMET), Sukkur, Pakistan, pp. 1-10.

[20] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints, Int. J. Comput. Vis., vol. 60, no. 2, pp. 91-110.

[21] Wang, Z. J. et al. (2020). CNN explainer: learning convolutional neural networks with interactive visualization, IEEE Trans. Vis. Comput. Graph., vol. 27, no. 2, pp. 1396-1406.

[22] Skuratov, V., Kuzmin, K., Nelin, I., & Sedankin, M. (2020). Application of a convolutional neural network and a Kohonen network for accelerated detection and recognition of objects in images, EUREKA: Phys. Eng., vol. 4, pp. 11-18.

[23] Tvoroshenko, I., & Tkachenko, D. (2020). Mechanisms of image classification based on descriptors of local features, in Proc. IV Int. Sci. Practic. Conf. «Integration of scientific bases into practice» (October 12-16, 2020), Stockholm, Sweden, pp. 443-448.

[24] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition, in Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, pp. 770-778.

[25] Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks, Int. J. Acad. Inf. Syst. Res., vol. 7, no. 7, pp. 25-36.

[26] Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). Image classification method modification based on model of logic processing of bit description weights vector, Telecommun. Radio Eng., vol. 79, no. 1, pp. 59-69.

[27] Liu, Y. H. (2018). Feature extraction and image recognition with convolutional neural networks, in J. Phys.: Conf. Ser., IOP Publishing, vol. 1087, no. 6, p. 062032.

[28] Gorokhovatsky, V. O., Pupchenko, D. V., & Solodchenko, K. G. (2018). Analysis of the properties, characteristics and results of using the latest detectors to determine specific points of the image, Control, Nav. Commun. Syst., vol. 1, p. 47. Гороховатський, В. О., Пупченко, Д. В., & Солодченко, К. Г. (2018). Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення, Системи управління, навігації та зв'язку, 1 (47), С. 93-98.

[29] Andersson, O., & Marquez, S. R. (2016). A comparison of object detection algorithms using unmanipulated testing images: Comparing SIFT, KAZE, AKAZE and ORB, pp. 1-28. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:927480/FULLTEXT01.pdf.

[30] Gorokhovatskyi, V., Peredrii, O., Tvoroshenko, I., & Markov, T. (2023). Distance matrix for a set of structural description components as a tool for image classifier creating. Adv. Inf. Syst., vol. 7, no. 1, pp. 5-13. Гороховатський, В. О., Передрій, О. О., Творошенко, І. С., & Марков, Т. Є. (2023). Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, Сучасні інформаційні системи, 7(1), С. 5-13.

[31] Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description, Adv. Electr. Electron. Eng., vol. 21, no. 1, pp. 19-27.

[32] Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for fast metric data search in structural methods for image classification, IEEE Access, vol. 10, pp. 124738-124746.

[33] Tvoroshenko, I., & Temchur, K. (2021). Features of software application development for food recognition using deep machine learning methods, Proc. XXVI Int. Sci. Practic. Conf. «Topical issues of practice and science», London, Great Britain, pp. 680-684.

[34] Couchot, J., Couturier, R., Guyeux, C., & Salomon, M. (2016). Steganalysis via a convolutional neural network using large convolution filters for Embedding Process with Same Stego Key, pp. 1-24. [Online]. Available: https://arxiv.org/pdf/1605.07946.pdf.

[35] Kabir, S., Islam, R. U., Hossain, M. S., & Andersson, K. (2020). An integrated approach of belief rule base and deep learning to predict air pollution. Sens., vol. 20, no 7, p. 1956.

[36] Ioffe, S. & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift, in Proc. 32nd Int. Conf. Mach. Learn., vol. 37, pp. 448-456.

[37] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res., vol. 15, pp. 1929-1958.

[38] McClure, N. (2017). TensorFlow machine learning cookbook. Packt Publishing Ltd.