Peoples' Friendship University of Russia

Faculty of Science

Foreign Languages Department

Development of iOS mobile application using machine learning technologies

Word count: 5010

Student: Leonova Alina

Group: НФИбд-02-17

Argument Consultant: Shorokhov S.G., Docent, Associate Professor

Style and Language Consultant: Kozhukhova Yu.V., Senior Lecturer

Moscow

2021

Abstract

At present, Machine Learning is one of the most popular approaches to accumulated data exploration. The importance of machine learning on mobile devices is constantly growing due to the great amount of data collected by means of mobile applications.

The work considers the capabilities of various libraries and frameworks that provide the implementation of machine learning technologies in Swift and Python programming languages. In addition, practical results of the work include the development of a mobile iOS-application for food image recognition in the Xcode environment. The study examines the entire technological chain from the machine learning model preparation and training to its usage in an iOS app.

The iOS platform was chosen due to the ability to effectively use the hardware features of Apple devices for the implementation of machine learning technologies.

Key words and phrases: machine learning, Python, Swift, iOS, application development tools.

Contents

Introduction

The objective of this study is to review Machine Learning technologies capabilities in mobile iOS applications (apps) implementation and to develop a mobile app for iOS using Machine Learning methods. The relevance of Machine Learning on mobile devices is constantly growing due to the great of data collected by means of mobile apps.

Machine Learning (ML) is an application of artificial intelligence (AI) that studies methods of building learning-capable algorithms. At present, ML is one of the most popular approaches to accumulated data exploration. Apple develops its own machine learning tools and also gives the opportunity to import and use some other machine learning models (Hollemans, 2018). Apple is already using machine learning in almost every aspect of the way we interact with our devices. It includes such features as sound and face detection, handwriting recognition and many others that make life easier for users. The company is actively improving this direction. They add new features to their apps and new development environment tools in each system update.

The aim of this work is to examine the entire technological chain from the machine learning model preparation and training to its usage in an iOS app.

The hypothesis of this work is that it is possible to create an app with ML features in iOS. The image classification task was used as an example of practical implementation.

The iOS platform and native iOS instruments were chosen for this work due to their ability to effectively use hardware features of Apple devices for the implementation of machine learning technologies. The development is based on the Xcode IDE (Integrated Development Environment, which is primary in iOS development). It can use Apple's machine learning tools. Most of them involve the use of third-party tools that allow you to import models previously generated in one of the supported languages. The foundation for using machine learning on the Apple platform is the Core ML framework, which runs on top of low-level CPU and GPU (Central and Graphics Processing Unit) optimized libraries and can use trained models prepared by other means.

1. Machine Learning Technologies in Modern Information Systems

1.1. Machine Learning objectives

1.1.1. Evolution of Machine Learning Approaches

The term "machine learning" (ML) was introduced in 1959 by Arthur Samuel, one of the pioneers of artificial intelligence (AI) research. By definition, ML is the “field of study that gives computers the ability to learn without being explicitly programmed”. Already in the early years of AI development, a number of researchers were actively engaged in the task of using data to train machines. This was also the time when Frank Rosenblatt developed the first perceptron, laying the foundation for neural network technology. The resurgence of interest in machine learning refers to the end of the 1990s, facilitated by the growing availability of digitized information and the development of telecommunications technology. This made it possible to formulate new goals for AI, focused on solvable problems of a practical nature.

Finally, the surge of practical interest in machine learning in recent decades has been influenced by the growing volume and variety of available data, combined with increased and cheaper computing power. As a consequence, software and hardware architectures for effective implementation of machine learning algorithms have been actively developed. At present, machine learning on large amounts of data has turned neural networks, which previously had rather limited capabilities, into an effective practical tool.

1.1.2. Classification of machine learning tasks

In general, machine learning refers to the direction in AI related to the construction of learning-capable algorithms.

The most common class of problems is **supervised learning** (learning with a teacher), which can be considered a generalization of the classical function approximation problem. In it, precedents form "object/response" pairs with some unknown dependence. The task of training in this case is to restore this dependence by constructing an algorithm that provides a sufficiently accurate classifying answer for any possible input object. The quality function is usually specified as the average error of the obtained answers for all objects in the sample. **Semi-supervised learning** combines the features of learning with and without a teacher. As in teacher-assisted learning, each precedent corresponds to an object/answer pair, but the answers are not known for all precedents. **Transductive inference** implies the need for prediction only for precedents from a test sample; it usually boils down to a partial learning task. **Learning with reinforcement** implies that for each precedent there is an object, a "situation/decision" pair. The responses are the values of the quality functional, which evaluates the effectiveness of the decision. Thus, the goal is to find a strategy of action in some environment to maximize the long-term gain (formation of investment strategies, self-learning of robots, automatic control of technological processes) (MachineLearning.ru, 2016).

1.2. Methods for solving machine learning problems

Machine learning has integrated a wide variety of approaches, those based on classical mathematical statistics and those of a more empirical nature. My work focuses on deep learning, which falls within the latter.

1.2.1. Deep Learning and Neural Networks

General principles. The concept of a neuron

Deep learning is a machine learning method using an artificial neural network that mimics the human brain, it consists of neurons organized into a network. Some neurons of the network (simple processors, see Figure 1) are labeled as network inputs and some as network outputs. The input vector is transformed into the output vector. This transformation is given by the weights of the network links, which are adjusted in the learning process. Thus, dependencies between input and output vectors are revealed, which makes the network capable of prediction (Geron, 2018).

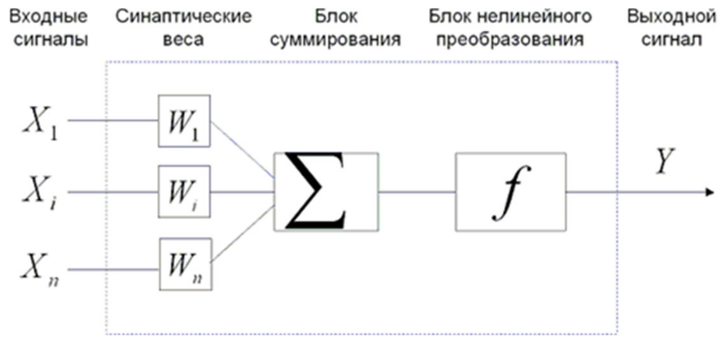


Figure 1 — Artificial neuron model

Types of networks

Neural networks consist of a huge number of connections between simple elements. The training is reduced to the construction of the optimal structure of connections and adjustment of parameters of these connections. Many neural network architectures thus far have been proposed, which can be classified according to the following criteria:

- by topology: fully connected, multi-layer, weakly connected;
- by the way of learning: with a teacher, without a teacher, with reinforcement;
- by model: direct propagation, recurrent (with feedbacks), based on radial basis functions, Kohonen networks, convolutional networks;
- by the way the weight coefficients are adjusted: with fixed and dynamic coefficients.

Let us consider a certain type of neural networks used in image processing.

Convolutional neural network

Convolutional neural network (CNN) have revolutionized pattern recognition and computer vision, but they have also been used successfully for other complex human tasks such as speech recognition and text analysis. CNN are often called the most successful model of deep learning.

The peculiarity of a convolutional neural network is its fundamental multilayeredness: neurons in it, as in the visual cortex, are grouped into layers of different types. Neurons of each layer have their own features, and only some of them are connected with the others from neighboring layers. Neurons in the first layers are divided into images of a certain size (also called maps), and within one layer different maps correspond to neurons of different types responding to different features of images.

Methods of determining the activation of the next layer in CNN are divided into two main types, corresponding to the two types of layers:

- in **convolutional** layers, the activation of neurons of the next layer is defined as a linear combination of the activations of neurons of the previous layer by multiplying the fragment by the convolution matrix; the scalar result of each convolution falls on some nonlinear activation function, formally representing the activation layer, which is usually logically combined with the convolutional layer;
- in **subsampling** layers, the activation of neurons of the next layer reproduces the activation of neurons of the previous layer, but the image size is reduced due to the pooling (subsampling or subsampling) procedure, which consists in replacing the activation of adjacent neurons with their maximum or their average.

The data coming out of the CNN is transmitted to the full-link layer (or full-link network, which can, in turn, consist of several layers), where the final classification of the most abstract concepts identified by the CNN from the original image (see Figure 2).

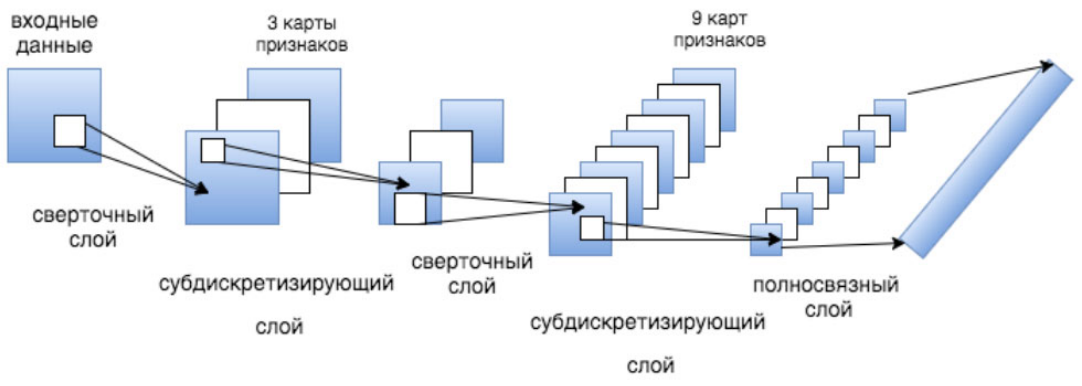


Figure 2 — An example of a CNN architecture

This structure of convolutional networks is very useful for working with images, because the network will get similar results for the images with slight visual differences.

1.3. Methods for developing cross-platform and mobile apps

The task of cross-platform development emerged from the natural desire of developers to minimize the effort required to create and maintain their software in the prevailing variety of hardware and system software. Cross-platform is realized with the use of tools providing portability of software at any given level.

The relevance and ways of solving this problem are changing with the appearance of new platforms or with the popularization and capture of dominant positions by certain platforms. For example, for a long time the Windows software platform in combination with Intel processors (even received the common name of Wintel) was the dominant platform for consumer systems, while at the same time for server systems the preference of developers was given to various Unix versions. The change in the situation in recent decades was significantly influenced by the emergence of mobile systems, the growing popularity of the Internet and "thin clients". As for the Russian developers, an import substitution course as well as the utilization of domestic platforms has also played a significant role in the development of events.

Despite the fact that cross-platform solutions simplify the development of platform-ready apps, they also have a number of drawbacks, which are the most acute in the case of mobile apps focused on using the most relevant platform features.

Firstly, all of them to some extent use virtualization tools or supporting libraries that hide operating system APIs (Application Programming Interface) from the app program code. This usually leads to an increase in CPU load and power consumption, i.e. a decrease in app performance and device autonomy.

What is more critical for using relevant platform features is that cross-platform mobile development tools inevitably limit app capabilities by providing a kind of averaged API, focused primarily on building a user interface.

In contrast to cross-platform, native solutions (written in a native programming language such as Java or Kotlin for Android and Objective-C or Swift for iOS), provide access to all functions of a given OS, allow arbitrary interface customization and are free of any performance issues. On the other hand, covering users of at least two popular platforms would require creating two separate apps, which would require more time, effort and resources (Yakimov, 2020).

Therefore, the most popular solution is the combined approach, in which cross-platform tools are used to implement the user interface, while the access to platform-specific features, such as the use of hardware-accelerated operations specific to machine learning, is provided by using separate components implemented with native tools.

2. Methods and tools for developing mobile apps that use machine learning

2.1. Machine Learning Technologies

The first step to start implement machine learning methods is a choice of a programming language. Therefore, current trends and comparative studies of the various languages' benefits in terms of current tasks were considered. As a result, it turned out, that according to the all recent rankings of programming languages, one of the leading positions is taken by Python. For example, IEEE Spectrum puts it on the first line of its ratings for recent years (IEEE, 2020).

Python is a high-level, dynamically typed and platform-independent open-source programming language. It was created in 1991 and is widely used for data analysis. Thanks to an active community, many different libraries and tools are developed for Python, including those for machine learning. Many of these libraries use hardware acceleration tools, primarily focused on GPUs (graphics gas pedals) and CUDA technology, in order to accelerate performance.

The following is a look at the most popular machine learning tools, their features and functionality, presumably suitable for my future work.

2.1.1. Python machine learning tools

Popular libraries

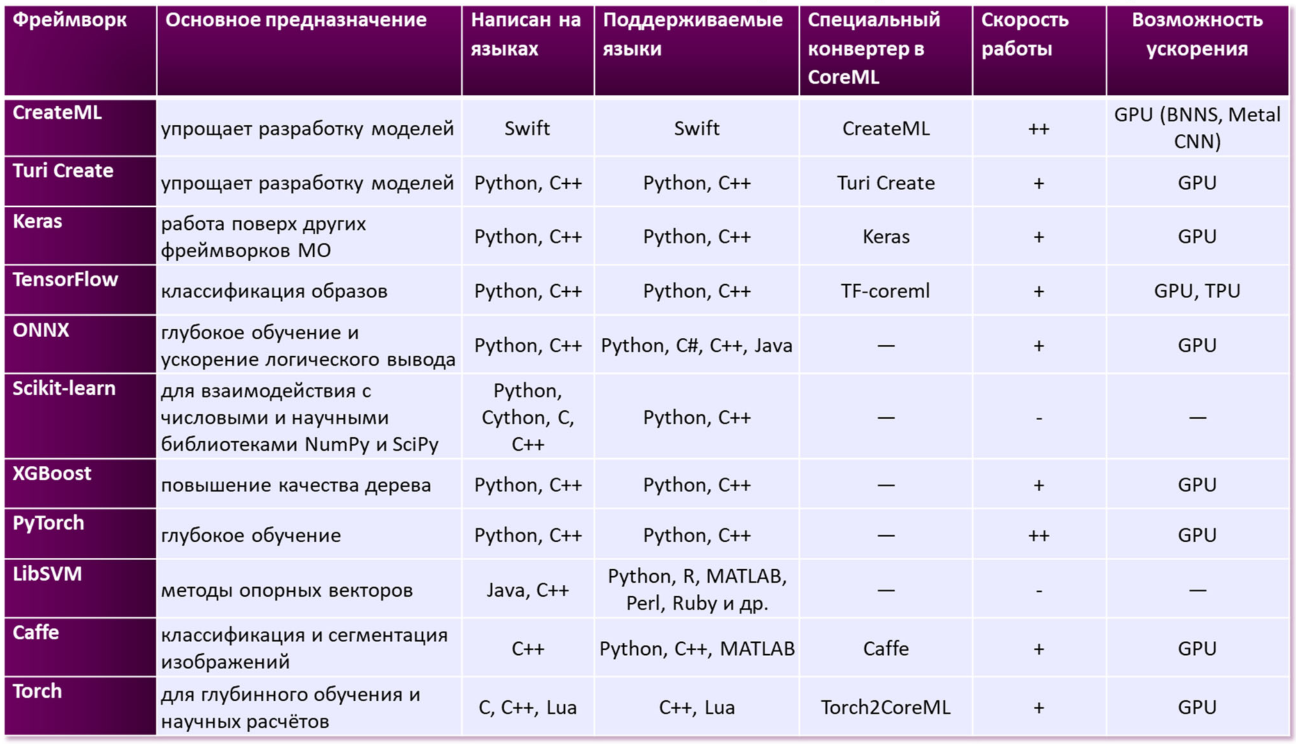
The features of the most popular frameworks were studied and assembled in Table 1 to justify the subsequent choice. It took into account the main purpose for which the framework was created, the programming languages in which it is written and which languages are supported. It also indicated that if there is a special converter for the framework in CoreML (more on this in the next section), perhaps they work better than CoreMLTools, which is universal for Python. Also, the speed of operation was considered in the table, which is, in principle, a consequence of having the ability to accelerate with GPUs.

Thus, when considering the available framework options, special attention was paid to the presence of implemented neural networks in them. As a result, it was decided to choose Keras and TensorFlow, mostly due to wide variety of the ready-made datasets, which are easy to experiment and work with immediately.

The main features of Keras and Tensorflow

Keras is an open source neural network library written in Python. It is an add-on to the frameworks Deeplearning4j, TensorFlow and Theano. TensorFlow is a Google Brain machine learning system, opened for free on November 9, 2015. TensorFlow can run on many parallel processors, both CPUs and GPUs, relying on the CUDA architecture to support general purpose computing on GPUs. With the release of TensorFlow 2.0, Google announced that it has converted Keras into the official high-level TensorFlow API.

Table 1— Comparison of machine learning frameworks



Фреймворк	Основное предназначение	Написан на языках	Поддерживаемые языки	Специальный конвертер в CoreML	Скорость работы	Возможность ускорения
CreateML	упрощает разработку моделей	Swift	Swift	CreateML	++	GPU (BNNS, Metal CNN)
Turi Create	упрощает разработку моделей	Python, C++	Python, C++	Turi Create	+	GPU
Keras	работа поверх других фреймворков МО	Python, C++	Python, C++	Keras	+	GPU
TensorFlow	классификация образов	Python, C++	Python, C++	TF-coreml	+	GPU, TPU
ONNX	глубокое обучение и ускорение логического вывода	Python, C++	Python, C#, C++, Java	—	+	GPU
Scikit-learn	для взаимодействия с числовыми и научными библиотеками NumPy и SciPy	Python, Cython, C, C++	Python, C++	—	-	—
XGBoost	повышение качества дерева	Python, C++	Python, C++	—	+	GPU
PyTorch	глубокое обучение	Python, C++	Python, C++	—	++	GPU
LibSVM	методы опорных векторов	Java, C++	Python, R, MATLAB, Perl, Ruby и др.	—	-	—
Caffe	классификация и сегментация изображений	C++	Python, C++, MATLAB	Caffe	+	GPU
Torch	для глубинного обучения и научных расчётов	C, C++, Lua	C++, Lua	Torch2CoreML	+	GPU

Popular neural network architectures used in pattern recognition

A pre-trained model is a model trained on some data set; it already contains weights and offsets that represent the features of the data set on which it has been trained. Using such models is useful because it saves time and computational resources.

Neural network architectures (InceptionV3, ResNet50, InceptionResNetV2)

Three implemented and pre-trained Keras networks were selected for further work: InceptionV3, ResNet50, InceptionResNetV2 (Keras, n.d.). The third section presents an analysis of their performance based on the construction of machine learning models on the selected dataset is presented.

There are different network architectures specific for different tasks. As for the image recognition, which is what my graduation paper concerns, Inception and ResNet architecture families are quite popular. The InceptionResNetV2 architecture combines some features of these families. Actually, they are implemented in Keras.

ResNet — Residual Network won ImageNet competition in 2015. In consisted of 152 layers and decreased error level to 3,57%. **InceptionV3** (GoogleNet V3) was suggested by Google in 2016 and is currently very popular for image processing. It consists mostly of convolution blocks. **InceptionResNetV2** is based on InceptionV3 with added residual blocks. Figure 3 shows its architecture (upper part — complete structure, lower part — compressed architecture).

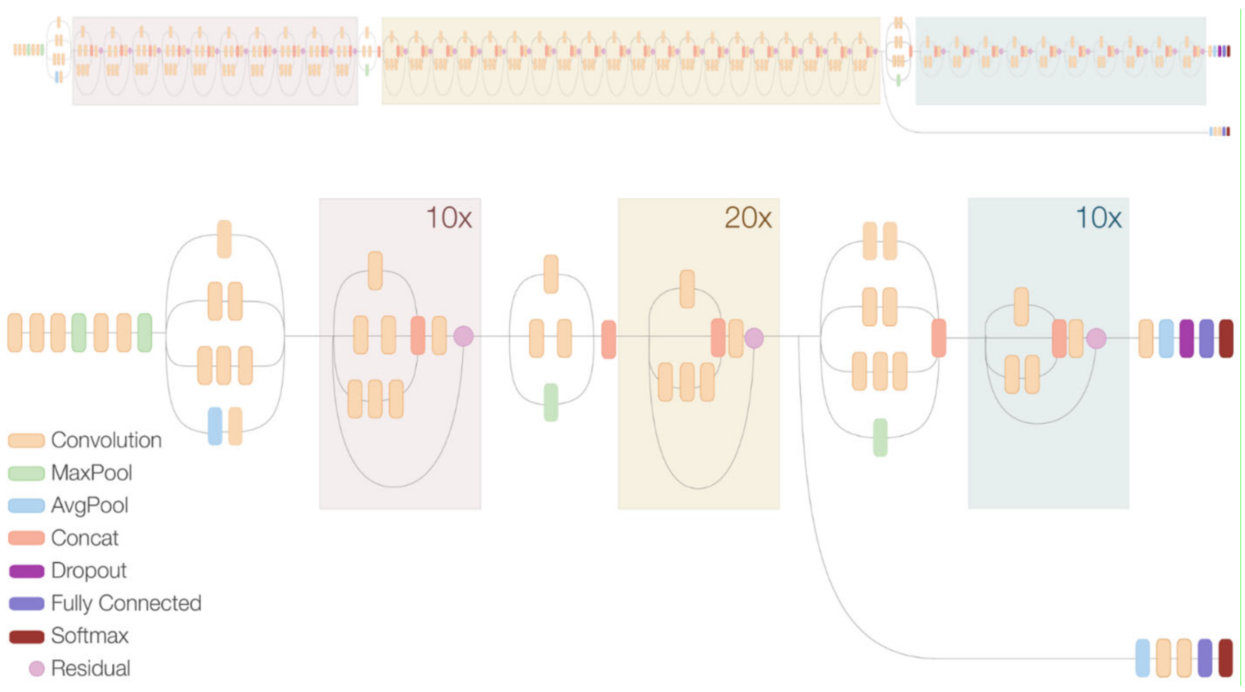


Figure 3 — InceptionResNetV2 network architecture

2.2. Machine Learning for Apple Devices

This work is dedicated to the creation of a mobile app on iOS that demonstrates the operation of a machine learning model. This requires exploring the existing capabilities that the company provides for importing and creating machine learning models.

On website for developers, Apple suggests two modules for implementing machine learning on its devices: Core ML and Create ML (Apple Inc., 2021). These modules work with subject-oriented frameworks (for image analysis, text processing, identification of sounds in audio). They use BNNS and Metal CNN libraries to optimize computation.

To implement machine learning inside an app, you need to:

- create a model (a file with `mlmodel` extension);
- add the model to Compile Sources, in order to compile it (make `mlmodelc`) every time the app is built and gives predictions as a result;
- use it in the app.

There are two basic approaches for creating models:

1. Import a pre-built and trained data model using one of the compatible frameworks (Rulin, 2018).

2. Create a model and select one of the existing templates (implemented in Xcode subject-oriented frameworks).

Let's take a closer look at the possible ways to implement machine learning in iOS app, shown schematically below (see Figure 4). All these ways lead to creating the Core ML model.

Core ML is the foundation for using machine learning on the Apple platform. This framework enables the usage of models derived from other popular frameworks such as TensorFlow, Keras, PyTorch, scikit-learn, Caffe. Apple's own products, Siri, Camera and QuickType, are based on it. Core ML makes it easy to integrate machine learning into an app and create different "smart" functions with a couple lines of code. It is also possible to encrypt the developed machine learning model using Xcode (Mishra, 2020).

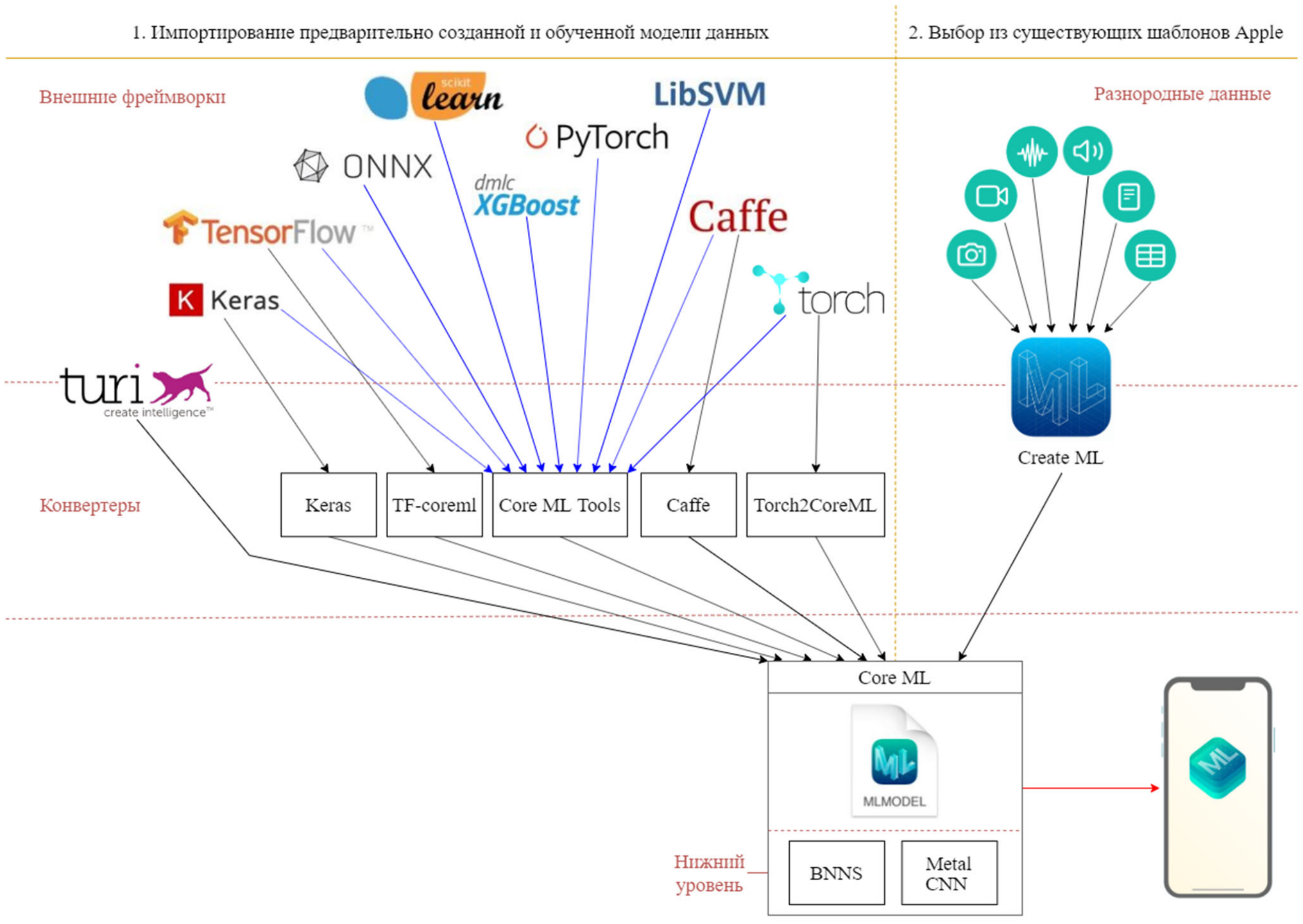


Figure 4 — Ways to implement machine learning in iOS apps

Core ML training without program code is possible because Create ML applies models that are already built into iOS (since version 12). This is a transfer learning — a method of rapidly learning models by reusing knowledge from another model trained earlier for a different task. In fact, it allows you to create new models by changing only the last layer of the neural network. Thus, this approach provides an easy model creation, decreasing both the time it takes the model to train and the disk space the resulting program occupies (Albon, 2019).

Core ML cannot accept data and train models on its own. It can only use some types of trained models, including those generated in Python, convert them into its own format, and make predictions. Core ML makes it easy to import various machine learning algorithms, such as tree ensembles, SVMs, and generalized linear models.

Most popular frameworks has converters of their models to Core ML. In addition to using models converted from popular cross-platform frameworks, some tools are focused directly on the Apple platform (Asteroid Technologies, 2018).

Create ML - Apple's own toolkit that allows you to train Core ML models without writing program code. To do this, Xcode has an interface to select one of 6 data types that the model will work with: images, video, activity (motion data from the accelerometer and gyroscope), sound, text and tables. Each data type has its own templates, there are 11 of them now (different classifiers, object detector, table regressor and others). Create ML also supports multi-model training (training of several models on different data sets in one project). You can control the training process: pause, save, resume and extend training. It is also possible to accelerate model learning using GPUs (Apple Inc., 2021).

All these toolkits are based on low-level APIs that work directly with the CPU and GPU:

BNNS (Basic Neural Network Subroutines), part of the Accelerate system, which is the basis for performing fast computations on the CPU.

Metal CNN (Convolutional Neural Networks) - part of the Metal Performance Shaders library, which optimizes the performance of cores and uses GPU instead of CPU (Hollemans, 2017).

2.3. Mobile app development tools

The most popular iOS app development environments or IDEs are Xcode from Apple and AppCode from JetBrains. **Xcode** has a fast editor and it equipped with a full set of development tools for iOS, macOS, etc. It can be downloaded from the App Store for free. **AppCode** also contains a full set of necessary tools for effective work with Objective-C, Swift, C++. It's more stable and faster than Xcode, but it lacks a visual debugger and you still need Xcode to build and run apps.

Since the aim of this study is to experiment with abilities and make a small app for graduation work, using AppCode in this situation is superfluous and Xcode was chosen.

Speaking of programming language for iOS development, you have to choose between Swift and Objective-C. **Objective-C** is a compiled object-oriented programming language, developed from the C language and is its superset. The language was created to solve the problem of code reuse. **Swift** was created by Apple primarily for iOS and macOS developers. Swift was conceived as an easier to read and more error-resistant language than its predecessor Objective-C. Programs in Swift are compiled using LLVM included in Xcode 6 and above. Swift can apply Objective-C runtime, which makes it possible to use both languages within one program (Wang, 2019).

Objective-C is still needed, because many projects still work in Objective-C. Moreover some Swift classes still contain Objective-C code. More and more developers prefer Swift, which means it's more quickly developed and in demand. It has been chosen in this work as well.

3. Development of a mobile app for image classification

Phases of app development are provided below.

The first phase of the work consists of preparing data, creating and training a machine learning model using `tensorflow.keras` in Python. Also, the learning process was accelerated by GPU operation.

The second phase of the work consists of importing and using the previously trained model and creating a mobile iOS app in Swift in Xcode environment in macOS Catalina operating system using VMware virtual machine (since Xcode requires MacOS to work).

3.1. Preparing and converting the model

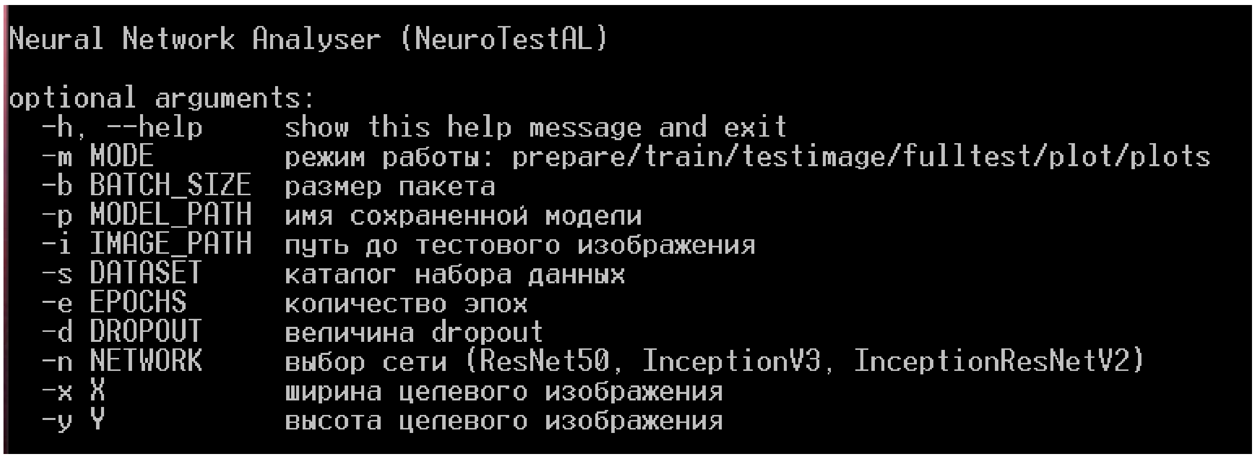
To implement the model, the work of neural networks ResNet50, InceptionV3, InceptionResNetV2, implemented in the package TensorFlow Keras Applications was compared.

3.1.1. Analysis of the neural network architectures' efficiency

To create a ML model for a mobile app, first, we had to choose which pre-trained model should be used. So the program NeuroTestAL was created for analysis of the efficiency of neural network architectures. The analysis was based on the machine learning models built on a tfds.image_classification.food101 dataset of images.

Three neural network architectures (ResNet50, InceptionV3, InceptionResNetV2) are considered in the program, but adding other networks to the program to analyze them can be easily done by importing the desired module and calling the program with the appropriate parameter.

The program contains several modes of operation, regulated by the startup parameters (Figure 5).



```
Neural Network Analyser (NeuroTestAL)

optional arguments:
  -h, --help     show this help message and exit
  -m MODE        режим работы: prepare/train/testimage/fulltest/plot/plots
  -b BATCH_SIZE  размер пакета
  -p MODEL_PATH  имя сохраненной модели
  -i IMAGE_PATH  путь до тестового изображения
  -s DATASET     каталог набора данных
  -e EPOCHS      количество эпох
  -d DROPOUT     величина dropout
  -n NETWORK     выбор сети (ResNet50, InceptionV3, InceptionResNetV2)
  -x X           ширина целевого изображения
  -y Y           высота целевого изображения
```

Figure 5 — Instructions with a description of all intended parameters

Since it is more convenient to apply the ready-to-use function `train_datagen.flow_from_directory` to load files from the specified directory with automatic breakdown by classes, we made preliminary preparation creating separate directories for training (train) and testing (test) and copying data into them. It happens in the `prepare` function.

The training mode of the model is set in the `train` function (see Listing 1). It makes data augmentation, specifies the parameters for model callbacks, creates the model and, finally, it calls `train_model` function.

```
def train(args):
    X_train, X_test = setup_generator(make_dataset_path(args, 'train'),
                           make_dataset_path(args, 'test'), args.batch_size, shape[:2])
    print(X_train)
    callbacks = []
    callbacks.append(ModelCheckpoint(filepath=make_model_path(args, args.dataset +
      '-weights.epoch-{epoch:02d}-val_loss-{val_loss:.4f}
      -val_accuracy-{val_accuracy:.4f}.hdf5'), verbose=1, save_best_only=True))
    callbacks.append(CSVLogger(make_model_path(args, 'history_log.csv')))
    model_final = create_model(args.network, X_train.num_classes, args.dropout, shape)
    train_model(model_final, X_train, X_test, callbacks, args)
```

Listing 1 — train function

As for the augmentation (see Listing 2), the ImageDataGenerator class was created to generate tensor image data packages (Keras, n.d.). It provides many arguments for regulating image changes and thereby creating additional training data. In order to achieve really good results, deep networks must be trained on a huge amount of data. Data augmentation is a technique for creating additional training data from existing data by making small changes to existing data. Such methods are of great help when working with the task of image classification. Small modifications of the image in the learning process does not spend a lot of resources, but as a result, significantly saves memory in comparison with how much information would need to be stored in the case of training on the same amount of data without augmentation. In addition, these methods are especially useful for the task of object recognition, where image markup is required before training.

```
def setup_generator(train_path, test_path, batch_size, dims):
    train_datagen = ImageDataGenerator(
        rotation_range=40,        width_shift_range=0.2,
        height_shift_range=0.2,   rescale=1. / 255,
        shear_range=0.2,          zoom_range=0.2,
        horizontal_flip=True,     fill_mode='nearest')
    test_datagen = ImageDataGenerator(rescale=1. / 255)
    train_generator = train_datagen.flow_from_directory(
        train_path, target_size=dims, batch_size=batch_size)
    validation_generator = test_datagen.flow_from_directory(
        test_path, target_size=dims, batch_size=batch_size)

    return train_generator, validation_generator
```

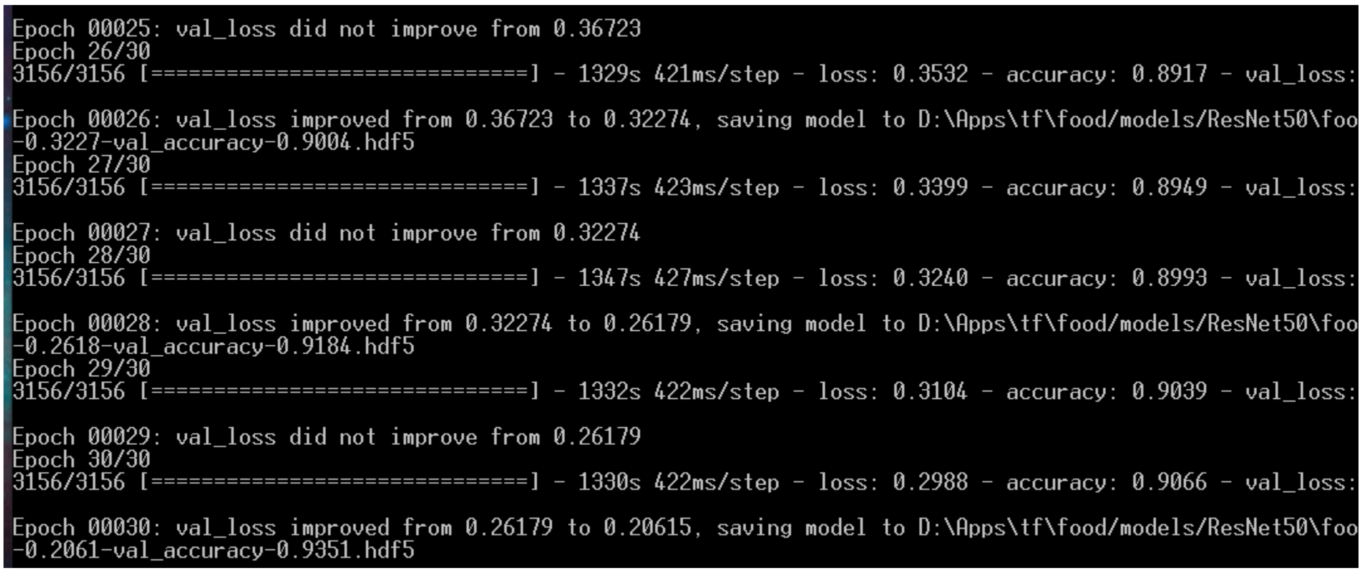
Listing 2 — setup_generator function

Finally, the training itself is performed by the `train_model` function (see Listing 3). First, the data are optimized and tuned by the `compile` method, which specifies a function that calculates losses (categorical cross-entropy) and a metric to evaluate the model - accuracy. After that the `fit_generator` method trains the model a specified number of times (epochs).

```
def train_model(model_final, train_generator, validation_generator, callbacks, args):
    model_final.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    model_final.fit_generator(train_generator,
        validation_data=validation_generator,
        epochs=args.epochs, callbacks=callbacks,
        steps_per_epoch=train_generator.samples//args.batch_size,
        validation_steps=validation_generator.samples//args.batch_size)
```

Listing 3 — train_model function

When you start this mode, model training begins. Each epoch takes 20-25 minutes and another 5 minutes to save. It was also noticed that around the 20th epoch, model readings were improving less and less frequently, so the default net epochs were reduced to 25 (see Figure 6).



```
Epoch 00025: val_loss did not improve from 0.36723
Epoch 26/30
3156/3156 [==============================] - 1329s 421ms/step - loss: 0.3532 - accuracy: 0.8917 - val_loss:

Epoch 00026: val_loss improved from 0.36723 to 0.32274, saving model to D:\Apps\tf\food/models/ResNet50\foo
-0.3227-val_accuracy-0.9004.hdf5
Epoch 27/30
3156/3156 [==============================] - 1337s 423ms/step - loss: 0.3399 - accuracy: 0.8949 - val_loss:

Epoch 00027: val_loss did not improve from 0.32274
Epoch 28/30
3156/3156 [==============================] - 1347s 427ms/step - loss: 0.3240 - accuracy: 0.8993 - val_loss:

Epoch 00028: val_loss improved from 0.32274 to 0.26179, saving model to D:\Apps\tf\food/models/ResNet50\foo
-0.2618-val_accuracy-0.9184.hdf5
Epoch 29/30
3156/3156 [==============================] - 1332s 422ms/step - loss: 0.3104 - accuracy: 0.9039 - val_loss:

Epoch 00029: val_loss did not improve from 0.26179
Epoch 30/30
3156/3156 [==============================] - 1330s 422ms/step - loss: 0.2988 - accuracy: 0.9066 - val_loss:

Epoch 00030: val_loss improved from 0.26179 to 0.20615, saving model to D:\Apps\tf\food/models/ResNet50\foo
-0.2061-val_accuracy-0.9351.hdf5
```

Figure 6 — Console output in progress (ending)

The `plot` and `plots` modes open the histopy_log.csv files and plot the accuracy and loss vs. epoch graphs. The `plot` mode does this for a particular selected model, and the `plots` mode plots all the models considered in the app.

The program was built and models were trained for all the 3 neural network architectures noted earlier. After that the program was ran in `plots` mode (see Figure 7). As an intermediate result of this work, InceptionResNetV2 gave the best results and was chosen to be imported into the iOS app.

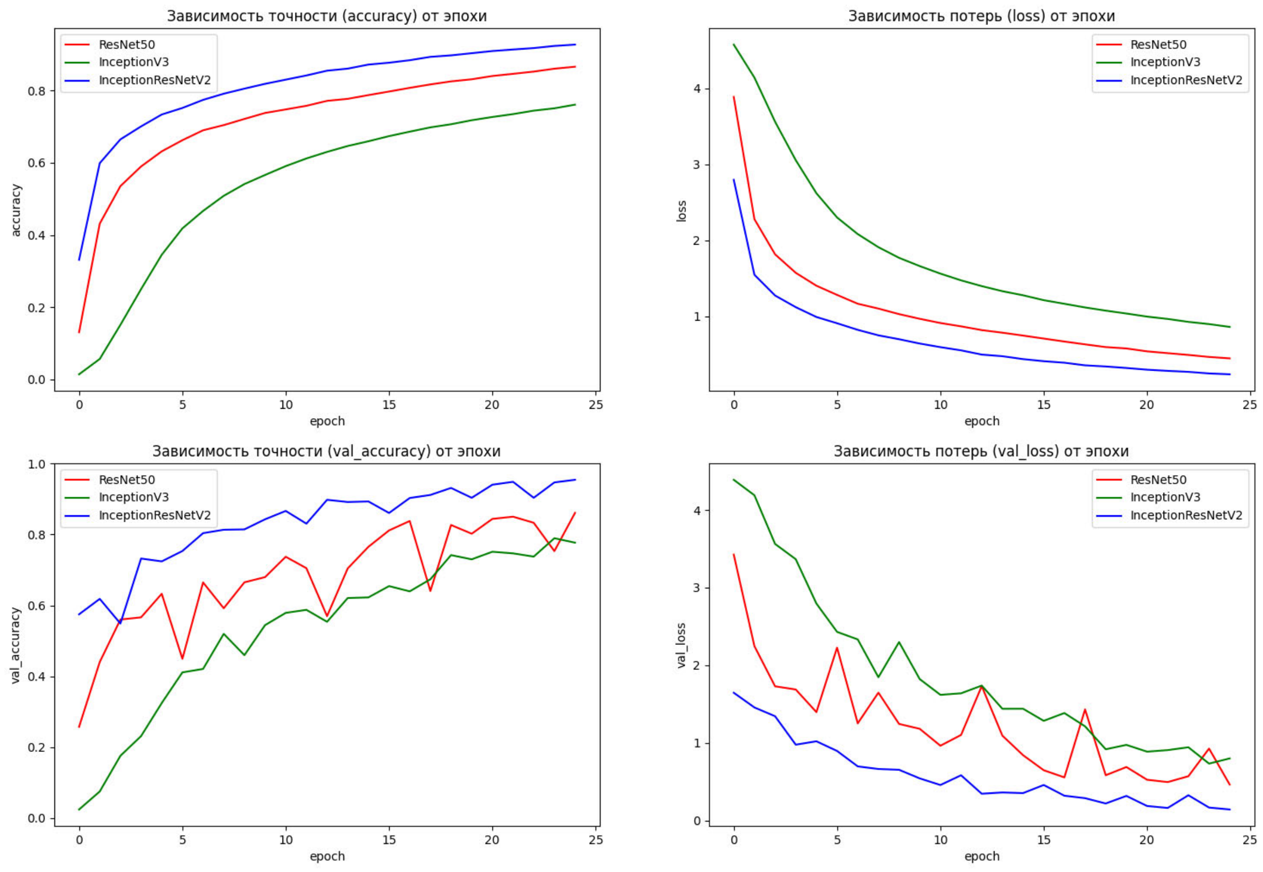


Figure 7 — Accuracy and loss vs. epoch graphs for all the networks (InceptionV3, ResNet50, InceptionResNetV2)

Two modes were implemented for testing. The test of a single image is performed by the `testimage` function, you must pass the path to the image being tested and specify the model. The `fulltest` function goes through the whole test.txt file, which contains the paths (class_name/image_name), for each of which the prediction by the selected model is performed.

3.2. Implementation of the app

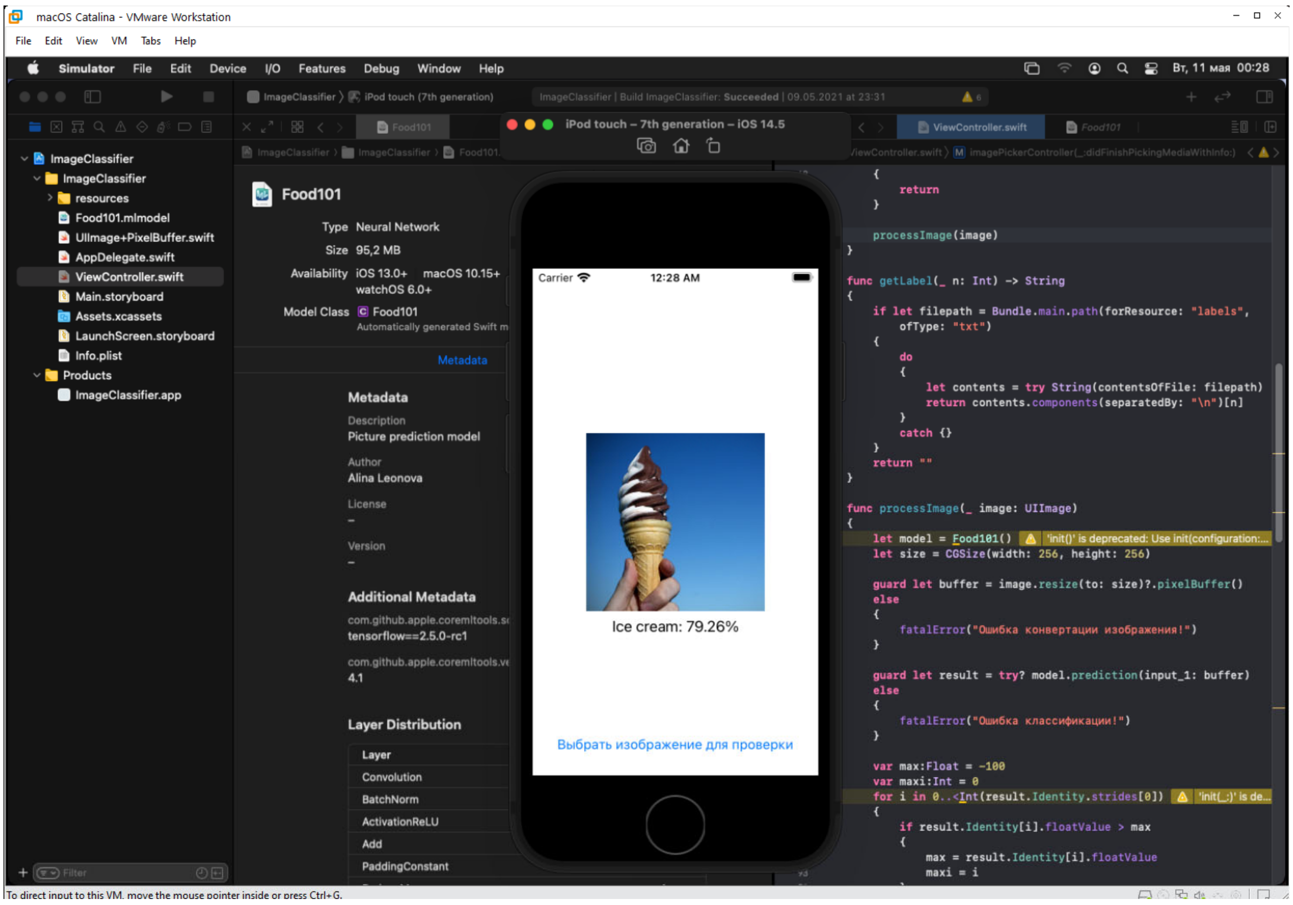
Script for converting the model to CoreM is presented in the Listing 4. All parameters of the future CoreML model are specified here. After running this script, the Food101.mlmodel appears in the active directory. Next mlmodel file moves to the app folder for using the model in the app.

```
import coremltools.converters as ct

mlmodel = ct.convert(model='food.h5', source="tensorflow", inputs=[ct.ImageType()])

mlmodel.author = 'Alina Leonova'
mlmodel.short_description = 'Picture prediction model'
mlmodel.input_description['input_1'] = 'Image'
mlmodel.output_description['Identity'] = 'Label'

mlmodel.save('Food101.mlmodel')
```

Listing 4 — convert.py file

So, to apply the CoreML model in the app code it is needed to import the CoreML module and initialize the previously imported model, then just use the model for predictions.

A screenshot in the Figure 8 shows: VMware macOS with opened Xcode, loaded app project in it, description of the imported model Food101, ViewController.swift file (which uses the imported model for the prediction) and the built-in Xcode iOS simulator demonstrating the work of the app (with a selected picture and a prediction it has an ice cream on it with 79.26% certainty).



macOS Catalina - VMware Workstation
File Edit View VM Tabs Help
Simulator File Edit Device I/O Features Debug Window Help
Вт, 11 мая 00:28
ImageClassifier | Build ImageClassifier: Succeeded | 09.05.2021 at 23:31
iPod touch – 7th generation – iOS 14.5
ImageClassifier
ImageClassifier
resources
Food101.mlmodel
UIImage+PixelBuffer.swift
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
ImageClassifier.app
Carrier
12:28 AM
Ice cream: 79.26%
Выбрать изображение для проверки
ViewController.swift
Food101
imagePickerController(_:didFinishPickingMediaWithInfo:)

Food101

Type Neural Network
Size 95,2 MB
Availability iOS 13.0+ macOS 10.15+ watchOS 6.0+
Model Class Food101
Automatically generated Swift m

Metadata

Description
Picture prediction model

Author
Alina Leonova

License
–

Version
–

Additional Metadata

com.github.apple.coremltools.s
tensorflow==2.5.0-rc1

com.github.apple.coremltools.v
4.1

Layer Distribution

Layer
Convolution
BatchNorm
ActivationReLU
Add
PaddingConstant

```
        {
            return
        }

        processImage(image)
    }

    func getLabel(_ n: Int) -> String
    {
        if let filepath = Bundle.main.path(forResource: "labels",
            ofType: "txt")
        {
            do
            {
                let contents = try String(contentsOfFile: filepath)
                return contents.components(separatedBy: "\n")[n]
            }
            catch {}
        }
        return ""
    }

    func processImage(_ image: UIImage)
    {
        let model = Food101()   'init()' is deprecated: Use init(configuration:...
        let size = CGSize(width: 256, height: 256)

        guard let buffer = image.resize(to: size)?.pixelBuffer()
        else
        {
            fatalError("Ошибка конвертации изображения!")
        }

        guard let result = try? model.prediction(input_1: buffer)
        else
        {
            fatalError("Ошибка классификации!")
        }

        var max:Float = -100
        var maxi:Int = 0
        for i in 0..<Int(result.Identity.strides[0])   'init(_:)' is de...
        {
            if result.Identity[i].floatValue > max
            {
                max = result.Identity[i].floatValue
                maxi = i
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Figure 8— Running the app in the built-in Xcode iOS simulator

Conclusion

This paper explored the use of machine learning technology in mobile app development.

An overview of machine learning methods, as well as their implementations in different Python frameworks, has been prepared. The study also involved a comparison of different neural networks architectures focused on working with image classification, and a conduction of models training and selection using an application written in Python using the Keros library of the Tensorflow framework.

The choice of native development tools for the iOS platform was justified and the available mobile APIs for machine learning were studied. The Swift language in the Xcode environment was chosen as the main development tool.

Finally, the hypothesis of this work was confirmed and the aim of the work was achieved. The practical results of the work include the development of the structure and a prototype of a mobile app with machine learning features. First, the ML model was created and trained with Python language, next, the model was converted to Core ML format and applied in the development of iOS app for food image recognition.

References

1. Albon, C. (2019). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning. ISBN: 9781491989388
2. Apple Inc. (2021). Create ML. Retrieved 1 April 2021, from https://developer.apple.com/machine-learning/create-ml/
3. Apple Inc. (2021). Machine Learning. Retrieved 1 April 2021, from https://developer.apple.com/machine-learning/
4. Asteroid Technologies. (2018). How to create CoreML models. Retrieved 1 April 2021, from https://medium.com/@asteroidzone1/how-to-create-coreml-models-366d43f3e438
5. Geron, A. (2018). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. ISBN: 9781492032649
6. Hollemans, M. (2017). Apple's deep learning frameworks: BNNS vs. Metal CNN. http://machinethink.net/blog/apple-deep-learning-bnns-versus-metal-cnn/
7. Hollemans, M. (2018). Machine Learning by Tutorials—Beginning Machine Learning for Apple and iOS. ISBN: 9781942878582
8. IEEE. (2020). Top Programming Languages 2020. Retrieved 1 April 2021, from https://spectrum.ieee.org/at-work/tech-careers/top-programming-language-2020
9. Keras. (n.d.). Image data preprocessing. Retrieved 1 April 2021, from https://keras.io/api/preprocessing/image/
10. Keras. (n.d.). Keras Applications. Retrieved 1 April 2021, from https://keras.io/api/applications/
11. MachineLearning.ru. (2016). Mashinnoe obuchenie [Machine Learning]. MachineLearning.Ru. Retrieved 1 April 2021, from http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение
12. Mishra, A. (2020). Machine Learning for iOS Developers. ISBN: 9781119602873
13. Rulin, R. (2018). Instrumenty Apple dlya mashinnogo obucheniya [Apple machine learning tools]. Retrieved 1 April 2021, from https://habr.com/ru/company/redmadrobot/blog/418307/
14. Wang. (2019). Pro iPhone Development with Swift 5 2e. ISBN: 9781484249444
15. Yakimov, S. (2020). Krossplatformennaya razrabotka mobil'nyh prilozhenij v 2020 godu [Cross-platform mobile app development in 2020]. Retrieved 1 April 2021, from https://habr.com/ru/post/491926/