Vol. 3, No. 1, 2021

DOI: https://doi.org/10.23939/csn2021.01.080
УДК 004.415.2

МЕТОДИ ТА ЗАСОБИ ПОКРАЩЕННЯ ТОЧНОСТІ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА МОБІЛЬНІЙ ПЛАТФОРМІ IOS В РЕАЛЬНОМУ ЧАСІ

Д. О. Кушнір

Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин
dimakush1@gmail.com

За результатами аналізу літературних джерел встановлено що перспективним напрямком пошуку та розпізнавання об'єктів є сімейство моделей Yolo. Проте існуючі реалізації не підтримують можливості запуску моделі на платформі iOS. Для досягення таких цілей розроблено комплексну масштабовану систему конвертації та покращення точності розпізнавання довільних моделей на базі системи Docker. Методика покращення полягає у додаванні до оригінальної моделі додаткового шару з функцією активації Mish. Методика конвертації полягає у оперативному перетовренні довільної моделі Yolo у формат CoreML. В рамках дослідження даних методик, була створена модель нейронної мережі Yolov4_TCAR. Додатково, розроблено метод акселерації навантаження на CPU при використанні додаткового шару нейронної мережі з функцією активації Mish на мові Swift під мобільну платформу iOS. В результаті, досліджено ефективність функції активації Mish, навантаження CPU мобільного пристрою, кількість використаної оперативної пам'яті та частоту кадрів при використані поліпшенної оригінальної моделі Yolov4-TCAR. Результати досліджень підтвердили функціонування алгоритму конвертації та покращення точості моделі нейронної мережі у реальному часі.

Ключові слова: Yolo, алгоритм конвертації та покращення вхідної моделі, модель нейронної мережі, функція активації, акселерація CPU, масштабована система, Mish, Docker, реальний час, Swift.

Вступ

Широке використання сучасних мобільних засобів, збільшення їх функціональних можливостей забезпечує розв'язання різних класів задач. Однією з них є пошук і розпізнавання певного класу об'єктів на зображенні чи відео потоці для подальшого його опрацювання. З розвитком мобільних пристроїв, спектр задач значно розширився, оскільки необхідно не тільки реалізувати модель розпізнавання, а й впровадити її у пристрій з меншими апаратними та програмними можливостями в порівнянні з обчисленням на сервері.

Огляд літературних джерел

На даний час існує багато сімейств моделей нейронних мереж, що надають можливість виконувати пошук та розпізнавання об'єктів на мобільній платформі [1]. Серед них важливо виділити сімейство моделей Yolo, які використовують розбивку вхідного відеопотоку на комірки, та вираховують ймовірності розпізнавання для кожної з них [2].

Загалом, можна виділити наступні моделі сімейства Yolo:

- *Yolov3 [2]* – створений на основі попередніх моделей Yolo із додаванням оцінки об'єктності до регіонів. Як backbone використовує фреймворк Darknet-53 замість ResNet-152 як у попередніх версіях. Як функцію активації використовує Relu [8]. Також до моделі була додана оцінка імовірностей на трьох окремих рівнях щоб покращити швидкість розпізнавання малих об'єктів.
- *Yolov4 [3]* – оновлена версія Yolo [2], яка показує поліпшеня в 10% mAP (mean Average Percision) в порівнянні з попередньою моделлю. Як backbone використовує модифіковану версію Darkned CSPDarknet53. Як фунцію активації використовує Mish [9]. Також використовується SPP(Spatial Pyramid Pooling) блок для підвищення ефективності рецептивного поля. В той же час блок PAN (Path Aggregation Network) використовується для більш ефетивної агрегації параметрів між різними рівнями backbone. Додатково, Yolov4 пропонує методи аугментації даних Mosaic та Self-Adversarial Training (SAT) для поліпшення процесу розпізнавання.
- *Yolov4 Scaled [4]* – модифікована версія yolov4 [3]. Має додатокві шари backbone: ResNet та ResNeXt та основний CSPDarknet53. Також добавлені деякий функціонал для масштабування потужностей моделі.
- *Yolov5 [5]* – Цілком нова імпелементація моделі Yolo [2] на фреймворку PyTorch. Тим не менше, при практично ідентичній архітектурі з Yolov4 [3], показує значне зменшення швидкості та якості розпізнавання зображень [7].
- *YoloR [6]* – Продовження дослідження в рамках підвищення ефективності моделі yolov4 [3]. Пропонується ідея добавити до механізму навчання не тільки свідоме пізнання (підготовані дані для навчання), але і несвідоме пізнання (за аналогіє з підсвідомістю людини).

Для подальших досліджень було обрано алгоритм Yolov4, оскільки він показав найбільшу ефективність серед усіх моделей нейронних мереж для мобільних пристроїв [7].

Визначено, що для поліпшення результатів роботи моделі нейронної мережі, необхідно добавити додатковий шар функцією активації. Для сімейства моделей Yolo, в залежності від їх архітектурної імплементації, можна застосувати наступні функції активації:

- *leaky ReLU [8]* – Функція активації, яка на відміну від своїх попередників є нелінійною. Повертає значення тільки якщо вхідне значення є додатнє. В цьому випадку Relu працює як ефективний аппроксіматор, що допомагає при необхідності об"єднати шарий нейронної мережі.
- *Mish [9]* – функція активації Mish, є модифікацією функції Relu [8]. При $x < 0$ є немонотонною функцією, при $x > 0$ аналогічна Relu. Вирішує проблему «помираючого Relu».
- *Softmax [10]* – функція, яка зазвичай використовується до вихідного шару задач класифікації. Вона гарантує, що сума всіх вихідних нейронів рівна 1, а значення інтервалу рівна [0, 1]. Використовується у функції активації Mish [9].
- *Sigmoid [11]* – степенева, нелінійна, не бінарна функція. На відміну від лінійних функцій, сигмоїда намагається привести значення змінною до однієї з її сторін (або в додатній площині, або у від'ємній) що корисно в задачах гласифікації зображень.Тим не менше, ця перевага є і недоліком: при наближені значення до кінця сигмоїди, градієнт змін приймає дуже малий діапазон можливих занчень. Що призводить до проблем з градієнтом зникнення.

Досліджено, що для портування такої моделі як Yolo на мобільний пристрій доцільно використати один з наступних фреймовorків машиного навчання: CoreML [12] чи TensorLite [13], які показали найліпші результати по продуктивності [14]. Вирішено обрати CoreML, оскільки ця система найбільше підходить для виконання задач під платформу iOS.

Для підвищення надійності та маштабованості системи конвертації та розгортання моделей Yolo на мобільному пристрої, було вирішено застосувати систему Docker [15]. Для даного класу задач Docker виявився найбільш ефективним, оскільки система не вимагає великих потужностей та обчислень, проте дає можливість налаштувати середовище та виконати необхідні операції.

Постановка завдання

На основі аналізу літературних джерел, реалізувати алгоритм конвертації моделі згорткової нейронної мережі Yolov4_TCAR у модель придатну для запуску на мобільній платформі iOS з форматом .mlpackage. Під час коневртації використати додатковий шар з функцією активації Mish для покращення отриманих результатів. Отриману модель використати в мобільному додатку, реалізованому на мові Swift для оперативного пошуку так розпізнавання об'єктів. Дослідити роботу системи у реальному часі та зробити висновки.

Запропонована схема, яка реалізує наведені вище задачі показана на рис 1.

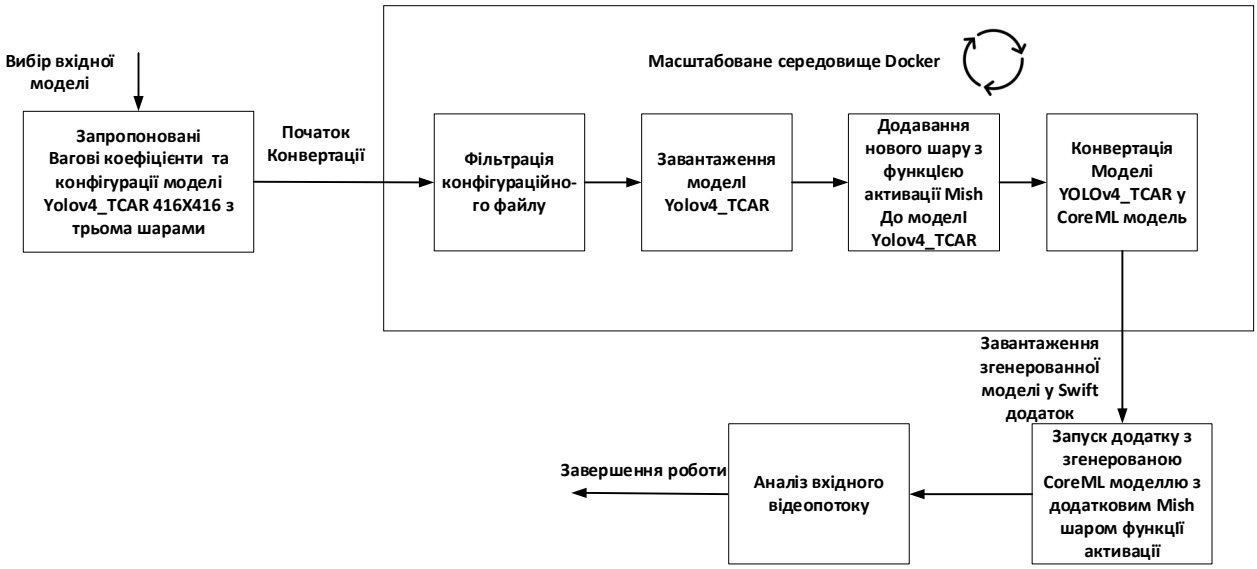

Рис. 1. Узагальнена структурна системи генерації та покращення точності розпізнавання моделі для мобільного додатку.

Як можна бачити на приведеній схемі, операції що виконують конвертацію моделі були огорнуті у систему масштабування Docker, для оперативного опрацювання будь якої кількості моделей різного виду у реальному часі.

Основні результати досліджень

Для вирішення поставленої задачі доцільно розділити її на три основні кроки:

Тренування моделі Yolov4_TCAR. Використовуючи згорткову нейронну мережу Yolov4 [3], та додаткові конфігураційні файли натренувати модель на розпізнавання специфічних об'єктів.

Конвератція моделі та додавання додаткового шару активації. Ця частина включає імплементацію алгоритмів конвертування моделі сімейства Yolov4 у модель для мобільних пристроїв CoreML. Для вирішення цієї задачі пропонується використати бібліотеку coremltools та середовище масштабування Docker. Для покращення точності результатів розпізанвання, під час конвератції пропнується добавити додатковий шар нейронної мережі з функцією активації Mish.

Реалізація моделі на мобільному додатку та тестування. У фінальній частині дослідження отримана модель нейронної мережі проходить крізь оригінальний шейдер Metal Mish для акселерації CPU мобільного пристрою. Були проаналізовані отримані результати опраюцвання відеозображень у реальному часі.

1. Тренування моделі Yolov4_TCAR

Модель Yolov4-512 на базі фреймоврку Darknet показала значні переваги, у порівнянн з прямими аналогами. Результати порівняння по FPS на базі данних COCO можна побачити на рис. 2.

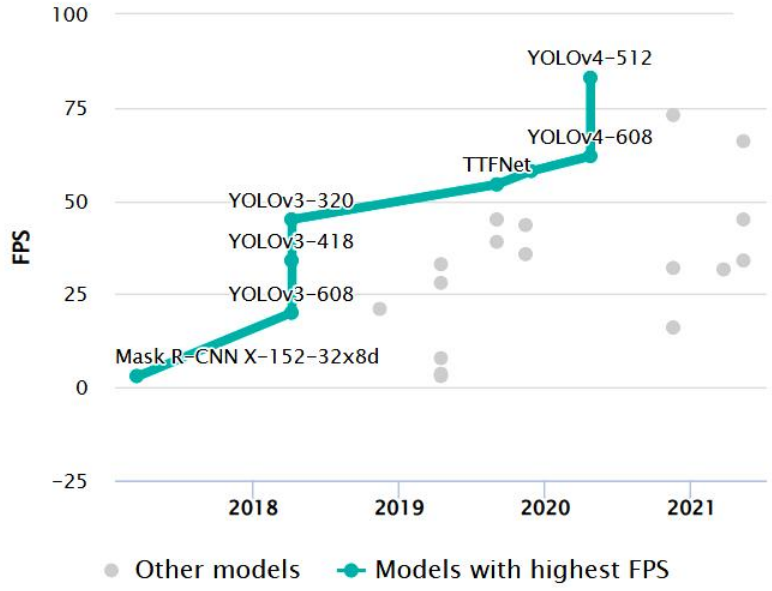

Рис. 2. Порівняння Yolov4-514 по FPS з іншими моделями нейронних мереж.
По осі x – рік створення моделі, по осі y – середня частота кадрів у секунду при розпізнавання.

Як видно з результатів дослідження, Yolov-512, станом на 2021 рік, підтримує FPS на рівні 85, що максимально наближає опрацювання відеопотоку до обробки в реальному часі.

В той же час *YOLOv4-TCAR* – запропонована реалізація моделі нейронної мережі. Аналогічно до Yolov4 має 3 вихідні шари, проте має додатковий шар активації Mish та натренована на меншу кількість об'єктів (всього три) і то му має перевагу у швидкості та розмірі.

2. Конвератція моделі та додавання додаткового шару активації

Для конвертації створенної у попередньому кроці моделі у формат CoreML було реалізовано масштабований сервіс опрацювання довільної кількості моделей. Сервіс реалізований на базі системи Docker та складається з двох частин: docker-compose (Лістинг 1) та DockerFile (Лістинг 2).

```
services:
   convertion-service:
    build: .
    image: tcar-network:convertion-service
    expose:  - '5000'
    networks: - tcar-network
    environment:
     models_data: ‘./${models_path}${model_name}’,
    volumes: ['./:/app']
```

Лістинг 1. Docker-compose файл системи конвертації

```
FROM python:3.7
COPY requirements.txt /app/requirements.txt WORKDIR /app RUN pip install -U pip RUN pip install -r requirements.txt
COPY . /app
ENTRYPOINT  sh ./prepare_config.sh ${MODEL_CFG_PATH} ${MODEL_CFG_TEMP_PATH} & \
        python  ./convert.py  -n  ${MODEL_CFG_PATH}  -c  ${MODEL_CFG_TEMP_PATH}  -w  ${MODEL_WEIGHTS_PATH}  -m
${MODEL_COREML_PATH}
```

Лістинг 2. DockerFile системи конвертації

Після отримання усіх необхідних параметрів, викликається сервіс фільтрації вхідних налаштувань конкретної моделі для стандартизації даних перед виконанням початкових операції конвертації.

За результатами аналізу літературних джерел, була реалізована функція активації Mish для спеціалізованого шару нейронної мережі. При порівнянні з іншими функціями активації, такими як Relu чи Sigmoid були отримані наступні результати (рис. 3).

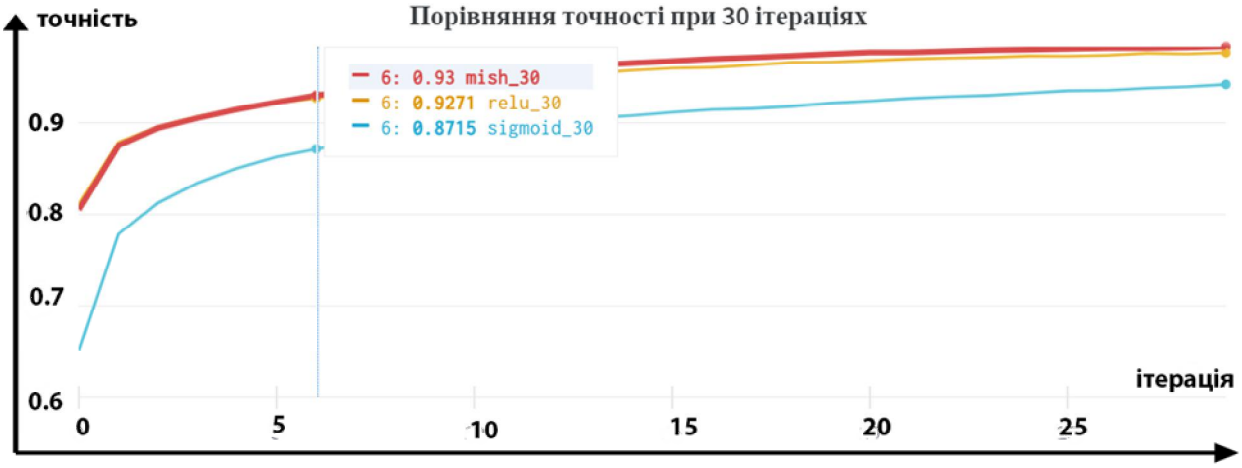

Рис. 3. Порівняння функції активації Mish (найвища у списку) з функціями Relu (посередині) та Sigmoid (нижча). По осі x – кількість ітерації, по осі y – точність розпізнавання.

Як видно, наприкінці тесту, точність тестування функції mish становила 0.982, що є кращим показником в порівнянні з функціями активаціх Relu та Sigmoid.

В загальному, функція активації Mish виглядає натсупним чином (рис. 4.).

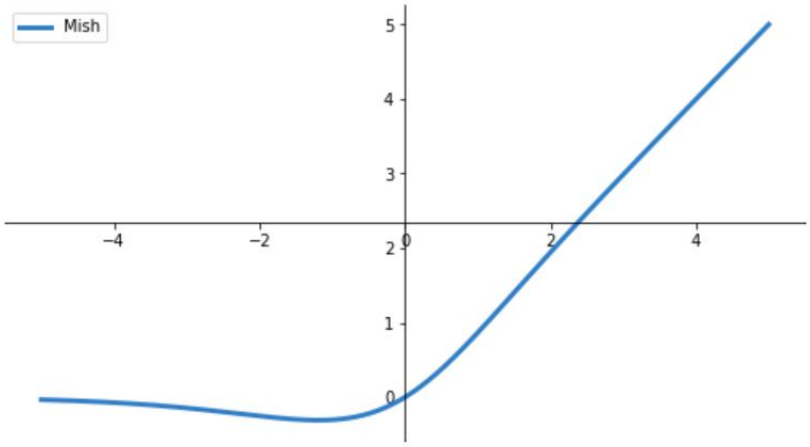

Рис. 4. Функція активації нейронної мережі Mish

Функцію Mish можна представити за допомогою наступної формули:

$$f(x) = x \cdot tanh(\varsigma(x)) \quad (1)$$

де, ς(x) – Softmax функція ln(1+e^x); x – вхідний вектор об'єктів розпізнавання.

Реалізована функція була додана до моделі нейронної мережі Yolov4_TCAR та зконвертована у Coreml модель (Лістинг 3).

```
# Оголошення класу Mish
class Mish(Layer):
    def __init__(self, **kwargs):
        super(Mish, self).__init__(**kwargs)
        self.supports_masking = True
    def call(self, inputs):
        return inputs * K.tanh(K.softplus(inputs))
    def get_config(self):
        config = super(Mish, self).get_config()
        return config
    def compute_output_shape(self, input_shape):
        return input_shape
# оголошення функції
def mish_convert_function(layer):
    args = NeuralNetwork_pb2.CustomLayerParams()
    args.className = "Mish"
    return args
# Конвертування моделі
    model  = Yolov4_TCAR()
    coreml_model = ct.converters.keras.convert(model,
                inputs=[ImageType(name='input_1', scale=1/255., color_layout="BGR",channel_first=False)],
                minimum_deployment_target=ct.target.iOS15,
                compute_precision=ct.precision.FLOAT16,
                add_custom_layers=True,custom_conversion_functions={ "Mish": mish_convert_function })
```

Лістинг 3. Лістинг функції активації Mish

В результаті була створена модель формату .mlpackage з 3 основними шарами та 1 додатковим шаром з функцією активації Mish. Для верифікації досліджень також була автоматично згенерована модель без додаткового шару з Mish.

3. Реалізація моделі на мобільному додатку

Модель була інтегрована у мобільний додаток для iOS, написаний на Swift 5. Для підвищення ефективності додаткового шару з функцією активації Mish, було реалізовано алгоритм акселерації CPU за допомогою функції mish.metal (Лістинг 4). Функція акселерації має зменшити навантаження на CPU піч час опрацювання додаткового шару нейронної мережі, оскільки функція Mish, не дивлячись на її переваги в точності, є реусурсозатратною.

```
#include <metal_stdlib>
using namespace metal;
kernel void mish(
  texture2d_array<half, access::read> inTexture [[texture(0)]],
  texture2d_array<half, access::write> outTexture [[texture(1)]],
  ushort3 gid [[thread_position_in_grid]])
{
  if (gid.x >= outTexture.get_width() || gid.y >= outTexture.get_height()) {
    return;
  }
  const float4 x = float4(inTexture.read(gid.xy, gid.z));
  const float4 y = x * tanh(log(1.0f + exp(x)));
  outTexture.write(half4(y), gid.xy, gid.z);
}
```

Лістинг 4. Функція акселерації Mish.metal

Як було згадано у попередній статті [16], після опрацювання кожного результуючого вектора, накладаються додаткові згладжучий та мінімізаційний фільтри для покращення результуючого масиву та відкидання лишніх чи невірно розпізнанних об’єктів.

4. Результати вимірювань

Відтестовано дві версії моделі Yolov4_TCAR з 3 шарами. В першій версії моделі був доданий спеціалізований шар з функцію активації Mish, друга версія складалась лише з базових 3 шарів. Результати вимірювань наведенні в таблиці.

Характеристика нейронних мереж

Модель нейронної мережі	**Тестований мобільний пристрій**	**Версія операційної системи системи**	**Середня Частота кадрів у секунду (FPS)**	**Середнє Навантаження на CPU (%) / кількість ядер процесора**	**Використання оперативної пам"ятті (MB)**
Yolov4_TCAR 416x416	Iphone 12	15.0.1	50	46 / 6	67
Yolov4_TCAR 416x416	Iphone 12	15.0.1	31	304 / 6	220
Yolov4_TCAR 416x416	Iphone 6s	12.5.5	15	89 / 2	77
Yolov4_TCAR 416x416	Iphone 6s	12.5.5	7	134 / 2	164

Як видно з результатів дослідження, додатковий шар з функцією активації Mish суттєво зменшує навантаження на процесор та оперативну пам’ять пристрою.

Висновки

У роботі запропоновано метод покращення точності оригінальної моделі нейронної мережі Yolov4_TCAR для мобільних пристроїв. Запропопнована методика полягає у додаванні додаткового шару до моделі нейронної мережі з функцію активації Mish. Ця функція активації показала найвищу точність (0.982) у порівнянні з прямими аналогами. Для реалізації функціоналу конвертації та покращення точності вхідних нейронних мереж була реалізована комлпексна масштабована система на базі Docker. Для зменшення навантаження на CPU мобільного пристрою, при використанні додатково шару з функцією активації Mish, був реалізований алгоритм акселерації Mish.metal. В результаті, був проведений набір тестів з використанням моделі з шаром з функцією активації Mish та без. Результати показали, що навантаження на процессор для моделі де використовується додатковий шар було зменшено в середньому у 2.5 рази, а частота кадрів збільшилась в середньому у 2 рази. Загалом, запропонований метод покращення точності оригінальної моделі нейронної мережі дозволив покращити якість розпізнавання у реальному часі.

1. Yuefeng Zhang, (2020). Deep Learning for Detecting Objects in an Image on Mobile Devices [Online]. Available: https://towardsdatascience.com/deep-learning-for-detecting-objects-in-an-image-on-mobile-devices-7d5b2e5621f9 (Accessed: April 2020)

2. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

3. Alexey Bochkovskiy, Joseph Redmon, Stefano Sinigardi, cyy, Tino Hager, JaledMC, Muhammad Maaz, Vinjn Zhang, Juuso Alasuutari, Philip Kahn, IlyaOvodov, Josh Veitch-Michaelis, Aymeric Dujardin, John Aughey, Akash Patel, duohappy, Aven, David Smith, Jud White, … Mosè Giordano. (2021). AlexeyAB/darknet: YOLOv4 (Version yolov4). Zenodo. DOI: https://doi.org/10.5281/zenodo.5622675

4. Kin-Yiu, Wong. (2021). Implementation of Scaled-YOLOv4 using PyTorch framework (v1.0.0). Zenodo. DOI: https://doi.org/10.5281/zenodo.5534091

5. Glenn Jocher, Alex Stoken, Ayush Chaurasia, Jirka Borovec, NanoCode012, TaoXie, Yonghye Kwon, Kalen Michael, Liu Changyu, Jiacong Fang, Abhiram V, Laughing, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Jebastin Nadar, imyhxy, Lorenzo Mammana, … wanghaoyang0106. (2021). ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support (v6.0). Zenodo. DOI: https://doi.org/10.5281/zenodo.5563715

6. Chien-Yao Wang, I-Hau Yeh, Hong-Yuan Mark Liao (2021). You Only Learn One Representation: Unified Network for Multiple Tasks [Online]. Available at: https://arxiv.org/abs/2105.04206 (Accessed: May 2021)

7. Chamidu Supeshala (2020). YOLO v4 or YOLO v5 or PP-YOLO? [Online]. Available at: https://blog.roboflow.com/yolov5-is-here (Accessed: June 2020)

8. Chaity Banerjee, Tathagata Mukherjee, and Eduardo Pasiliao. 2020. The Multi-phase ReLU Activation Function. In Proceedings of the 2020 ACM Southeast Conference (ACM SE '20). Association for Computing Machinery, New York, NY, USA, 239–242. DOI:https://doi.org/10.1145/3374135.3385313

9. Diganta Misra (2019). Mish: A Self Regularized Non-Monotonic Activation Function [Online]. Available at: https://arxiv.org/abs/1908.08681 (Accessed: June 2020)

10. Joshi, V., Das, A., Sun, E., Mehta, R.R., Li, J., Gong, Y. (2021) Multiple Softmax Architecture for Streaming Multilingual End-to-End ASR Systems. Proc. Interspeech 2021, 1767-1771, doi: 10.21437/Interspeech.2021-1298

11. Sridhar Narayan (1997). The generalized sigmoid activation function: Competitive supervised learning [Online]. doi: https://doi.org/10.1016/S0020-0255(96)00200-9 (Accessed: June 1997)

12. Abhishek Mishra. "Machine Learning for iOS Developers", John Wiley & Sons, 2020. DOI: 10.1002/9781119602927

13. Li Shuangfeng. TensorFlow Lite: On-Device Machine Learning Framework[J]. Journal of Computer Research and Development, 2020, 57(9): 1839-1853. DOI: https://doi.org/10.7544/issn1000-1239.2020.20200291

14. Mateusz Opala (2018). TensorLite. Core ML vs TensorflowLite: ML Mobile Frameworks Comparison [Online]. Available at: https://www.netguru.com/blog/coreml-vs-tensorflow-lite-mobile (Accessed: December 2018)

15. Dirk Merkel (2014). "Docker: lightweight Linux containers for consistent development and deployment". Linux journal, 2014, No. 239, –pp.2 [online] Available at: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment (Accessed: May 2014)

16. D. Kushnir and Y. Paramud, "Model for Real-Time Object Searching and Recognizing on Mobile Platform," 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 2020, pp. 127-130, doi: 10.1109/TCSET49122.2020.235407.

METHODS AND MEANS FOR REAL-TIME OBJECT RECOGNITION ACCURACY INCREASE IN VIDEO IMAGES ON IOS MOBILE PLATFORM

D. Kushnir

Lviv Polytechnic National University,
Computer Engineering Department

As a result of the analytical review, it was established that the family of Yolo models is a promising area of search and recognition of objects. However, existing implementations do not support the ability to run the model on the iOS platform. To achieve these goals, a comprehensive scalable conversion system has been developed to improve the recognition accuracy of arbitrary models based on the Docker system. The method of improvement is to add a layer with the Mish activation function to the original model. The method of conversion is to quickly convert any Yolo model to CoreML format. As part of the study of these techniques, a model of the neural network Yolov4_TCAR was created. Additionally, a method of accelerating the load on the CPU using an additional layer of neural network with the function of activating Mish in Swift for the iOS mobile platform was added. As a result, the effectiveness of the Mish activation function, the CPU load of the mobile device, the amount of RAM used, and the frame rate when using the improved original Yolov4-TCAR model were studied. The results of the research confirmed the functioning of the algorithm for conversion and accuracy increase of the neural network model in real-time.

Key words: **Yolo, input model conversion and improvement algorithm, neural network model, activation function, CPU acceleration, scalable system, Mish, Docker, real time, Swift.**