ДОДАТОК А
ГРАФІЧНИЙ МАТЕРІАЛ АТЕСТАЦІЙНОЇ РОБОТИ

Харківський національний університет радіоелектроніки
Кафедра АПОТ

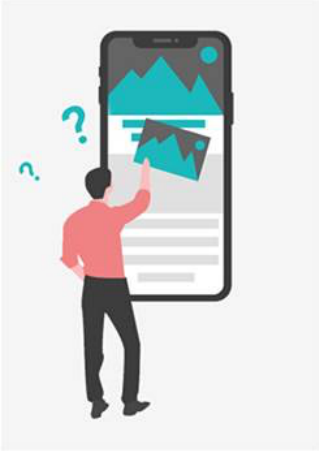
Атестаційна робота магістра

Спеціалізовані комп'ютерні засоби для розпізнавання контурних зображень

Виконав: ст. гр. СКСм-20-2 Погрібний Євгеній Євгенович
Керівник: проф. Кривуля Геннадій Федорович

Мета роботи

Метою даної роботи є розробка програмного забезпечення для розпізнавання контурних зображень, що дозволяє виділити контур на зображенні та розпізнати об'єкт на фото.



Постановка задачі

Для реалізації поставленої мети необхідно розробити мобільний додаток, який буде мати такі можливості:

- Зробити новий знімок або обрати зображення з галереї для подальшого аналізу
- Виділити контур на зображенні
- Розпізнати класифікацію зображеного об’єкта
- Відобразити результат

Загальні поняття

- Контурний аналіз – це сукупність методів виділення, опису та перетворення контурів зображення, де контур – це межа об’єкта, сукупність точок (пікселів), що відокремлюють об’єкт від фону.

- Розпізнавання образів – це розділ теорії штучного інтелекту, що вивчає методи класифікації об’єктів. За традицією об’єкт, що піддається класифікації, називається *образом*.

Основні технології

- Computer Vision – це область штучного інтелекту, яка дозволяє комп'ютерам і системам отримувати значущу інформацію з цифрових зображень, відео та інших візуальних даних.

- Machine Learning – це галузь штучного інтелекту та комп'ютерних наук, яка зосереджена на використанні даних та алгоритмів для імітації способу навчання людей, поступово покращуючи його точність.

Використовувані програмні засоби

- Середовище розробки Android Studio 
- Мова програмування Kotlin 
- Бібліотека OpenCV 
- Бібліотека TensorFlow Lite 

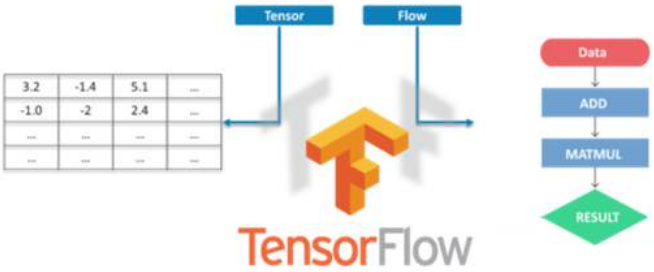
Бібліотека OpenCV для виділення контуру на зображенні

OpenCV – це бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом, розроблена компанією Intel, яка надає засоби для обробки і аналізу вмісту зображень

Бібліотека TensorFlow Lite для розпізнавання об’єктів на зображенні

TensorFlow Lite – це мобільна бібліотека для розгортування моделей на мобільних, мікроконтролерних та інших периферійних пристроях.

TensorFlow – платформа, яка дозволяє легко створювати та розгортати моделі машинного навчання.



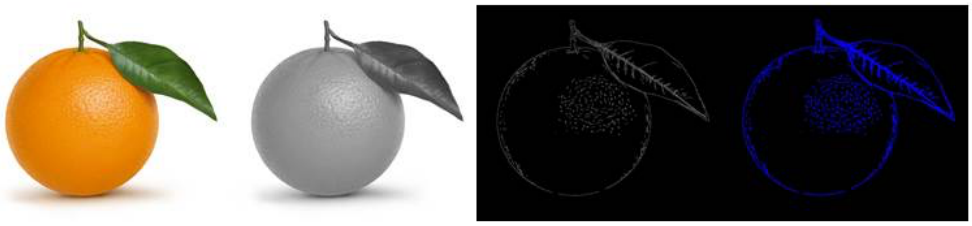
Алгоритм Канні

Алгоритм Кенни складається з п’яти окремих кроків:

- Згладжування зображення для видалення шуму
- Пошук градієнта – межі виділяються там, де градієнт зораження набуває максимального значення
- Придушення «Не максимумів» – тільки локальні максимуми відзначаються як межі
- Подвійна порогова фільтрація – потенційні межі виділяються порогами
- Трасування області неоднозначності – придушення всіх меж, що не зв’язані з «сильними» межами

Процес виділення контуру на зображенні

- Перетворення зображення у формат відтінків сірого
- Розмивання зображення для зменьшення шумів
- Застосування алгоритму Канні
- Знаходження контурів
- Відображення результату

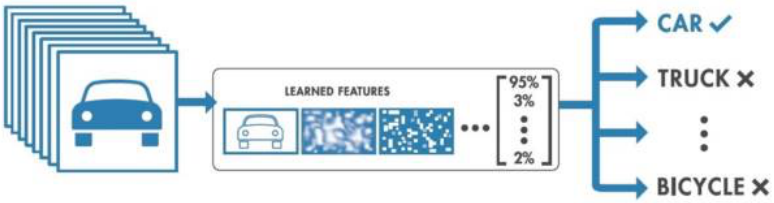


Програмна реалізація алгоритму виділення контурів на зображенні

```
private fun findContours(originalMat: Mat, checked: Boolean) {
    val finalMat = Mat(originalMat.rows(), originalMat.cols(), originalMat.type())
    val tempImage = Mat()
    val contours = ArrayList<MatOfPoint>()
    val hierarchy = Mat()
    val color = Scalar( v0: 255.0,  v1: 0.0,  v2: 0.0)

    Imgproc.cvtColor(originalMat, tempImage, Imgproc.COLOR_BGR2GRAY)
    Imgproc.blur(tempImage, tempImage, Size( width: 4.0,  height: 4.0))
    Imgproc.Canny(tempImage, tempImage,  threshold1: 40.0,  threshold2: 60.0)
    Imgproc.findContours(tempImage, contours, hierarchy, Imgproc.RETR_TREE, Imgproc.CHAIN_APPROX_SIMPLE)
    val drawMat = if (checked) originalMat else finalMat
    Imgproc.drawContours(drawMat, contours,  contourIdx: -1, color,  thickness: 4)
    showMatImage(drawMat)
}
```

Розпізнавання зображених об’єктів

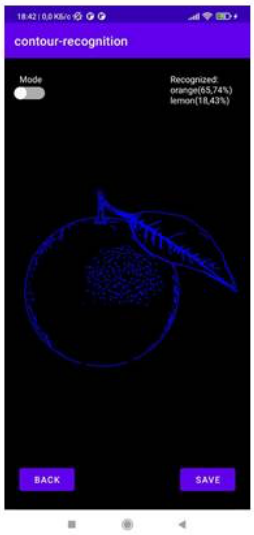


- Навчання та підготовка моделі
- Перетворення зображення на буфер байтів
- Передача зображення моделі для розпізнавання
- Конвертація результату в читабельний вигляд

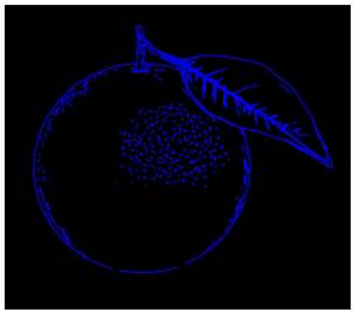
Програмна реалізація розпізнавання зображених об’єктів

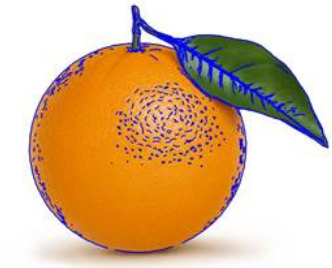
```
fun recognizeImage(bitmap: Bitmap): List<RecognizeResult> {
    val scaledBitmap = Bitmap.createScaledBitmap(bitmap, inputSize, inputSize,  filter: false)
    val byteBuffer = convertBitmapToByteBuffer(scaledBitmap)
    return if (quant) {
        val result = Array( size: 1) {  it: Int
            ByteArray(labelList.size)
        }
        interpreter?.run(byteBuffer, result)
        getSortedResultByte(result)
    } else {
        val result = Array( size: 1) {  it: Int
            FloatArray(labelList.size)
        }
        interpreter?.run(byteBuffer, result)
        getSortedResultFloat(result)
    }
}
```

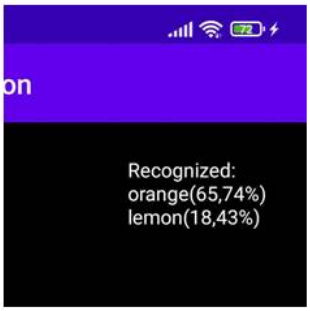
Інтерфейс мобільного додатку



Результат роботи мобільного додатку







Висновки

- Проаналізовано способи виділення контурів
- Досліджено методи розпізнавання зображень
- Розроблено мобільний додаток для виділення контуру та розпізнавання об’єктів на зображенні
- Додаток розроблено для ОС Android на мові програмування Kotlin
- Для пошуку контурів була використана бібліотека OpenCV
- Розпізнавання об’єктів реалізовано за допомогою бібліотеки TensorFlow Lite


Дякую за увагу

ДОДАТОК Б

Програмна реалізація проекту

CameraFragment.kt

```
class CameraFragment : Fragment() {

    private val openImage = registerForActivityResult(
        ActivityResultContracts.GetContent()) { uri: Uri? ->
        findNavController().navigate(CameraFragmentDirections
            .actionCameraFragmentToImageFragment(uri))
    }

    private val viewModel: CameraViewModel by viewModels()
    private lateinit var binding: FragmentCameraBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = DataBindingUtil.inflate(
            inflater, R.layout.fragment_camera, container, false)

        binding.openImageButton.setOnClickListener {
            openImage.launch("image/*")
        }

        binding.cameraViewfinder.setLifecycleOwner(viewLifecycleOwner)
        binding.cameraViewfinder.addCameraListener(object:
CameraListener() {
            override fun onPictureTaken(result: PictureResult) {
                super.onPictureTaken(result)
                findNavController().navigate(CameraFragmentDirections
                        .actionCameraFragmentToImageFragment(imageBitmap
= TakePictureResult(result))
                )
            }
        })

        binding.takePhotoButton.setOnClickListener {
            if (binding.cameraViewfinder.isTakingPicture)
                return@setOnClickListener
            binding.cameraViewfinder.takePicture()
        }

        return binding.root
    }

    override fun onRequestPermissionsResult(requestCode: Int,
permissions: Array<out String>, grantResults: IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
        val valid = grantResults.all { it == PERMISSION_GRANTED }
        if (valid && !binding.cameraViewfinder.isOpened) {
            binding.cameraViewfinder.open()
        }
    }
}
```

ImageFragment.kt

```
class ImageFragment : Fragment() {

    private val args:ImageFragmentArgs by navArgs()
    private val viewModel: ImageViewModel by viewModels()
    private lateinit var binding: FragmentImageBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = DataBindingUtil.inflate(inflater,
R.layout.fragment_image, container, false)
        binding.lifecycleOwner = viewLifecycleOwner
        binding.viewModel = viewModel

        context?.assets?.let { viewModel.initImageRecognize(it) }

        args.imageUri?.let {

viewModel.findContoursInImage(PathUtil.copyFileToInternal(context, it),
false)
            context?.contentResolver?.let { resolver ->
viewModel.recognizeImage(resolver, it) }
        }
        args.imageBitmap?.let {
            it.pictureResult.toBitmap { bitmap ->
                viewModel.findContoursInImage(bitmap)
                viewModel.recognizeImage(bitmap)
            }

        }

        binding.backButton.setOnClickListener {
            findNavController().navigateUp()
        }

        binding.imageModeSwitch.setOnCheckedChangeListener {
                _, checked -> args.imageUri?.let {
                    viewModel.findContoursInImage(
                     PathUtil.copyFileToInternal(context, it),
                     checked)
            }
        }

        return binding.root
    }
}
```

ImageViewModel.kt

```
class ImageViewModel : ViewModel() {

    private val viewModelJob = Job()
    private val coroutineScope = CoroutineScope(Dispatchers.Main +
viewModelJob)

    @Inject
    lateinit var imageRecognizeManager: ImageRecognizeManager

    private val _imageBitmap = MutableLiveData<Bitmap>()
```

```
        val imageBitmap: LiveData<Bitmap> get() = _imageBitmap

        private val _recognizeResults =
MutableLiveData<List<RecognizeResult>>()
        val recognizeResults: LiveData<List<RecognizeResult>> get() =
_recognizeResults

        init {
            Application.appComponent.injectImageViewModel(this)
        }

        fun initImageRecognize(assetManager: AssetManager) {
            imageRecognizeManager.init(assetManager)
        }

        fun recognizeImage(contentResolver: ContentResolver, uri: Uri) {
            val bitmap = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P)

ImageDecoder.decodeBitmap(ImageDecoder.createSource(contentResolver, uri))
                    .copy(Bitmap.Config.ARGB_8888, true)
            else MediaStore.Images.Media.getBitmap(contentResolver, uri)

            recognizeImage(bitmap)
        }

        fun recognizeImage(bitmap: Bitmap?) {
            _recognizeResults.value = bitmap?.let {
imageRecognizeManager.recognizeImage(it) }
        }

        fun findContoursInImage(imagePath: String, checked: Boolean) {
            coroutineScope.launch {
                findContours(imagePath, checked)
            }
        }

        fun findContoursInImage(bitmap: Bitmap?) {
            coroutineScope.launch {
                bitmap?.let { findContours(it) }
            }
        }

        private fun findContours(imagePath: String, checked: Boolean) {
            Log.i("ImageBitmapCheck", "image path = $imagePath")
            val originalMat = Imgcodecs.imread(imagePath)
            findContours(originalMat, checked)
        }

        private fun findContours(bitmap: Bitmap) {
            val originalMat = Mat()
            Utils.bitmapToMat(bitmap, originalMat, false)
            findContours(originalMat, false)
        }

        private fun findContours(originalMat: Mat, checked: Boolean) {
            val finalMat = Mat(originalMat.rows(), originalMat.cols(),
originalMat.type())
            val tempImage = Mat()
            val contours = ArrayList<MatOfPoint>()
            val hierarchy = Mat()
            val color = Scalar(255.0, 0.0, 0.0)

            Imgproc.cvtColor(originalMat, tempImage, Imgproc.COLOR_BGR2GRAY)
            Imgproc.blur(tempImage, tempImage, Size(4.0, 4.0))
```

```
            Imgproc.Canny(tempImage, tempImage, 40.0, 60.0)
            Imgproc.findContours(tempImage, contours, hierarchy,
Imgproc.RETR_TREE, Imgproc.CHAIN_APPROX_SIMPLE)
            val drawMat = if (checked) originalMat else finalMat
            Imgproc.drawContours(drawMat, contours, -1, color, 4)
            showMatImage(drawMat)
        }

        private fun showMatImage(mat: Mat) {
            Imgproc.cvtColor(mat, mat, Imgproc.COLOR_BGR2RGB)
            val bitmap = Bitmap.createBitmap(mat.width(), mat.height(),
Bitmap.Config.ARGB_8888)
            Utils.matToBitmap(mat, bitmap, true)
            _imageBitmap.value = bitmap
        }

        private fun closeImageRecognize() {
            imageRecognizeManager.close()
        }

        override fun onCleared() {
            super.onCleared()
            viewModelJob.cancel()
            closeImageRecognize()
        }

    }
```

RecognizeResult.kt

```
    data class RecognizeResult(
        val id: String,
        val title: String,
        val confidence: Float,
        val quant: Boolean
    ) {
        override fun toString(): String {
            return "$title${String.format("(%.2f%%)", confidence * 100)}"
        }
    }
```

Відомість кваліфікаційної роботи

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК. 46741Х.002 ПЗ	Пояснювальна записка	76 с.
		Графічні документи	
2		Слайд-презентація	17 слайдів
		Носії інформації	
3		Файл пояснювальної записки	1 шт.

					ГЮІК. 46741Х.002 ВД	
Змін.	Арк	Номер докум.	Підп.	Дата		
Розроб.		Погрібний Є. Є.			Спеціалізовані комп’ютерні засоби для розпізнавання контурних зображень Відомість кваліфікаційної роботи	Літ: У; Аркуш; Аркушів
Перевір.		Кривуля Г. Ф.				
Н.контр.		Шкіль О.С.				ХНУРЕ Кафедра АПОТ
Затв.		Чумаченко С.В.				