**SOURCE CODE**

|  |
| --- |
| private const val REQUEST\_PERMISSIONS = 1 |
|  | private const val REQUEST\_TAKE\_PICTURE = 2 |
|  |  |
|  | class MainActivity : AppCompatActivity() { |
|  |  |
|  | private var photoFilePath = "" |
|  |  |
|  | override fun onCreate(savedInstanceState: Bundle?) { |
|  | super.onCreate(savedInstanceState) |
|  | setContentView(R.layout.activity\_main) |
|  |  |
|  | checkPermissions() |
|  | } |
|  |  |
|  | private fun checkPermissions() { |
|  | if (arePermissionAlreadyGranted()) { |
|  | takePhoto() |
|  | } else { |
|  | requestPermissions() |
|  | } |
|  | } |
|  |  |
|  | private fun arePermissionAlreadyGranted() = |
|  | ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE\_EXTERNAL\_STORAGE) == PackageManager.PERMISSION\_GRANTED |
|  |  |
|  | private fun requestPermissions() { |
|  | ActivityCompat.requestPermissions(this, |
|  |  |
|  |  |
|  | arrayOf(Manifest.permission.WRITE\_EXTERNAL\_STORAGE), |
|  | REQUEST\_PERMISSIONS) |
|  | } |
|  |  |
|  | private fun takePhoto() { |
|  | photoFilePath = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY\_PICTURES).absolutePath + "/${System.currentTimeMillis()}.jpg" |
|  | val currentPhotoUri = UriHelper.getUriFromFilePath(this, photoFilePath) |
|  |  |
|  | val takePictureIntent = Intent(MediaStore.ACTION\_IMAGE\_CAPTURE) |
|  | takePictureIntent.putExtra(MediaStore.EXTRA\_OUTPUT, currentPhotoUri) |
|  | takePictureIntent.flags = Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION |
|  |  |
|  | if (takePictureIntent.resolveActivity(packageManager) != null) { |
|  | startActivityForResult(takePictureIntent, REQUEST\_TAKE\_PICTURE) |
|  | } |
|  | } |
|  |  |
|  | override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) { |
|  | if (requestCode == REQUEST\_PERMISSIONS && arePermissionGranted(grantResults)) { |
|  | takePhoto() |
|  | } else { |
|  | requestPermissions() |
|  | } |
|  | } |
|  |  |
|  | private fun arePermissionGranted(grantResults: IntArray) = |
|  | grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION\_GRANTED |
|  |  |
|  | override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) { |
|  | val file = File(photoFilePath) |
|  | if (requestCode == REQUEST\_TAKE\_PICTURE && file.exists()) { |
|  | //classify photo |
|  | } |
|  | } |
|  | } |
| class MainActivity : AppCompatActivity() { |
|  |  |
|  | private lateinit var classifier: Classifier |
|  | //other properties |
|  |  |
|  | override fun onCreate(savedInstanceState: Bundle?) { |
|  | super.onCreate(savedInstanceState) |
|  | setContentView(R.layout.activity\_main) |
|  |  |
|  | checkPermissions() |
|  | } |
|  |  |
|  | private fun checkPermissions() { |
|  | if (arePermissionAlreadyGranted()) { |
|  | init() |
|  | } else { |
|  | requestPermissions() |
|  | } |
|  | } |
|  |  |
|  | private fun init() { |
|  | createClassifier() |
|  | takePhoto() |
|  | } |
|  |  |
|  | private fun createClassifier() { |
|  | classifier = ImageClassifierFactory.create( |
|  | assets, |
|  | GRAPH\_FILE\_PATH, |
|  | LABELS\_FILE\_PATH, |
|  | IMAGE\_SIZE, |
|  | GRAPH\_INPUT\_NAME, |
|  | GRAPH\_OUTPUT\_NAME |
|  | ) |
|  | } |
|  |  |
|  | //... |
|  | }  |  | | --- | | class MainActivity : AppCompatActivity { | |  |  | |  | //... | |  | private val handler = Handler() | |  | private var photoFilePath = "" | |  |  | |  | override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) { | |  | val file = File(photoFilePath) | |  | if (requestCode == REQUEST\_TAKE\_PICTURE && file.exists()) { | |  | classifyPhoto(file) | |  | } | |  | } | |  |  | |  | private fun classifyPhoto(file: File) { | |  | val photoBitmap = BitmapFactory.decodeFile(file.absolutePath) | |  | val croppedBitmap = ImageUtils.getCroppedBitmap(photoBitmap) | |  | classifyAndShowResult(croppedBitmap) | |  | imagePhoto.setImageBitmap(photoBitmap) | |  | } | |  |  | |  | private fun classifyAndShowResult(croppedBitmap: Bitmap) { | |  | runInBackground( | |  | Runnable { | |  | val result = classifier.recognizeImage(croppedBitmap) | |  | //show result | |  | }) | |  | } | |  |  | |  | @Synchronized | |  | private fun runInBackground(runnable: Runnable) { | |  | handler.post(runnable) | |  | } | |  | } |  |  | | --- | | object ImageClassifierFactory { | |  |  | |  | fun create( | |  | assetManager: AssetManager, | |  | graphFilePath: String, | |  | labelsFilePath: String, | |  | imageSize: Int, | |  | inputName: String, | |  | outputName: String | |  | ): Classifier { | |  |  | |  | val labels = FileUtils.getLabels(assetManager, labelsFilePath) | |  |  | |  | return ImageClassifier( | |  | inputName = inputName, | |  | outputName = outputName, | |  | imageSize = imageSize.toLong(), | |  | labels = labels, | |  | imageBitmapPixels = IntArray(imageSize \* imageSize), | |  | imageNormalizedPixels = FloatArray(imageSize \* imageSize \* COLOR\_CHANNELS), | |  | results = FloatArray(labels.size), | |  | tensorFlowInference = TensorFlowInferenceInterface(assetManager, graphFilePath) | |  | ) | |  | } | |  | } |  |  | | --- | | private const val ENABLE\_LOG\_STATS = false | |  |  | |  | class ImageClassifier ( | |  | private val inputName: String, | |  | private val outputName: String, | |  | private val imageSize: Long, | |  | private val labels: List<String>, | |  | private val imageBitmapPixels: IntArray, | |  | private val imageNormalizedPixels: FloatArray, | |  | private val results: FloatArray, | |  | private val tensorFlowInference: TensorFlowInferenceInterface | |  | ) : Classifier { | |  |  | |  | override fun recognizeImage(bitmap: Bitmap): Result { | |  | preprocessImageToNormalizedFloats(bitmap) | |  | classifyImageToOutputs() | |  | val outputQueue = getResults() | |  | return outputQueue.poll() | |  | } | |  |  | |  | private fun preprocessImageToNormalizedFloats(bitmap: Bitmap) { | |  | // Preprocess the image data from 0-255 int to normalized float based | |  | // on the provided parameters. | |  | } | |  |  | |  | private fun classifyImageToOutputs() { | |  | //feed the classifier with the data via input | |  | tensorFlowInference.feed(inputName, imageNormalizedPixels, | |  | 1L, imageSize, imageSize, COLOR\_CHANNELS.toLong()) | |  | //run the classification | |  | tensorFlowInference.run(arrayOf(outputName), ENABLE\_LOG\_STATS) | |  | //get the results from the ouptput | |  | tensorFlowInference.fetch(outputName, results) | |  | } | |  |  | |  | private fun getResults(): PriorityQueue<Result> { | |  | val outputQueue = createOutputQueue() | |  | results.indices.mapTo(outputQueue) { Result(labels[it], results[it]) } | |  | return outputQueue | |  | } | |  |  | |  | private fun createOutputQueue(): PriorityQueue<Result> { | |  | return PriorityQueue( | |  | initialCapacity = labels.size, | |  | Comparator { (\_, rConfidence), (\_, lConfidence) -> | |  | Float.compare(lConfidence, rConfidence) }) | |  | } | |  | } |  **HOW IT SHOWS THE RESULT…..**  |  | | --- | | class MainActivity : AppCompatActivity { | |  |  | |  | //... | |  |  | |  | private fun classifyAndShowResult(croppedBitmap: Bitmap) { | |  | runInBackground( | |  | Runnable { | |  | val result = classifier.recognizeImage(croppedBitmap) | |  | showResult(result) | |  | }) | |  | } | |  |  | |  | @Synchronized | |  | private fun runInBackground(runnable: Runnable) { | |  | handler.post(runnable) | |  | } | |  |  | |  | private fun showResult(result: Result) { | |  | textResult.text = result.result.toUpperCase() | |  | layoutContainer.setBackgroundColor(getColorFromResult(result.result)) | |  | } | |  |  | |  | @Suppress("DEPRECATION") | |  | private fun getColorFromResult(result: String): Int { | |  | return if (result == getString(R.string.hot)) { | |  | resources.getColor(R.color.hot) | |  | } else { | |  | resources.getColor(R.color.not) | |  | } | |  | } | |  | } | |