УДК 004

Разработка мобильного приложения с нейросетью для решения задач пользователя

Андриенко Иван Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье представлена разработка мобильного приложения с нейросетью. Используется Gemini API, который предоставляет доступ к нейронным сетям Google Cloud. Приложение позволяет пользователям отправлять текст или изображения нейросети, а затем получать результаты обработки. Описан процесс разработки приложения, включая создание пользовательского интерфейса, работу с API Google Cloud и реализацию основных функций приложения. Приложение может быть использовано для различных целей, таких как перевод языков, распознавание объектов, поиск информации и т.д.
Ключевые слова: Мобильное приложение, Android, нейронные сети, Gemini API, распознавание текста, распознавание изображений, Android Studio.

Development of a mobile application with a neural network for solving user tasks

Andrienko Ivan Sergeevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article presents the development of a mobile application with a neural network. Gemini API is used, which provides access to Google Cloud neural networks. The application allows users to send text or images to the neural network, and then receive the processing results. The application development process is described, including creating a user interface, working with the Google Cloud API and implementing the main functions of the application. The application can be used for various purposes such as language translation, object recognition, information retrieval, etc.
Keywords: Mobile application, Android, Neural Networks, Gemini API, Text recognition, Image recognition, Android Studio.

1 Введение

1.1 Актуальность

Мобильные приложения становятся все более сложными и функциональными по мере того, как технологии развиваются. Они выполняют все больше задач, становятся более персонализированными и удобными для

пользователей. Использование нейросетей в мобильных приложениях позволяет расширить их возможности и сделать их более полезными и удобными для пользователей. Gemini — это семейство мультимодальных больших языковых моделей, разработанных Google DeepMind. Он был анонсирован 6 декабря 2023 года и позиционировался как претендент на GPT-4 от OpenAI.

1.2 Обзор исследований

В своей работе В.А. Острицова рассмотрела возможность применения нейронных сетей в качестве нового источника на пути к прогрессу в разработке мобильных приложений, описан принцип их действия и проанализирован современный мобильный рынок [1]. К.А. Кравченко, А.С.Щутский, А.С. Креймер проанализировали нейронную сеть на основе архитектуры CNN для определения рецепта блюда по изображению. Анализ изображений блюд с использованием нейронных сетей потенциально дает возможность определять их примерный состав и возможный рецепт приготовления. Предложена концепция мобильного приложения для проведения такого анализа путем передачи изображения на сервер, который производит анализ с использованием нейронной сети. Рассмотрена возможность загрузки обучающих изображений и рецептов [2]. В своей работе А.С. Исаченков, Г.П. Эльсессер, О.В. Гречкина провели задачу конвертации нейронной сети, разработанной с использованием библиотеки Keras в формат Tensorflow Lite, её перенос на мобильное устройство под управлением Android, и использование нейронной сети в приложении. Было разработано мобильное приложения для Android, которое позволяет загружать изображения из файловой системы устройства, передавать их на обработку нейронной сети, получать и выводить результат классификации на экран [3]. В своей работе Д.А. Калашникова рассмотрела архитектуры нейронных сетей с возможностью идентификации объектов, реализованные с использованием библиотеки TensorFlow для мобильных приложений [4]. В статье Е.Е. Ануфриева рассматривает применение алгоритмов нейронных сетей и искусственного интеллекта в мобильных приложениях. В работе дан краткий анализ на проблему использования нейронных сетей в определении психологического состояния человека и прогнозировании его дальнейших действий для улучшения своего самочувствия. Приведен анализ приложений, в которых используется искусственный интеллект, составлены преимущества и недостатки использования машинного обучения в роли психолога и психиатра [5].

1.3 Цель исследования

Цель исследования – Разработать мобильное приложение с нейросетью, использующее Gemini API, способное решать разные задачи пользователя.

2 Материалы и методы

Процесс создания мобильного приложения проделан в среде Android Studio, с использованием языка программирования Kotlin и Java.

3 Результаты и обсуждения

На первом этапе разработки мобильного приложения с использованием Gemini API, необходимо создать новый проект в среде разработки Android Studio. Важным этапом является выбор шаблона, исключительно Gemini API Starter, который предоставляет базовую структуру проекта и интеграцию с API. Данный шаблон проекта доступен только на бета-версии Studio. В качестве sdk используется минимальный из доступных (рис. 1).

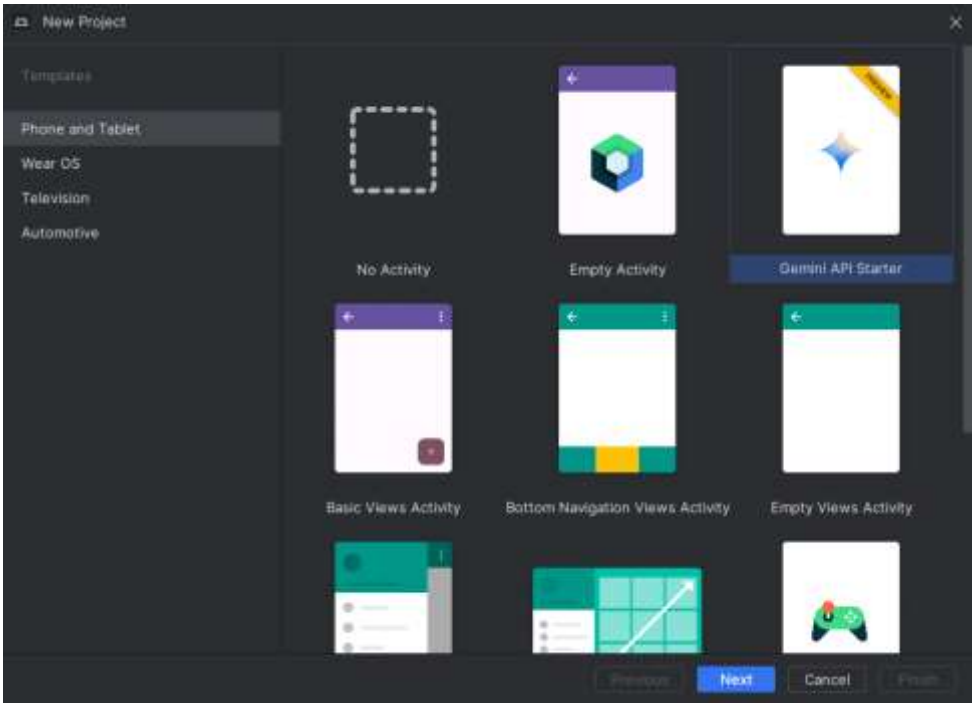

Рисунок 1 – Выбор базового шаблона

При создании проекта у разработчика запрашивают api ключ. Данный ключ можно получить на сайте Google Al Studio [6]. Базовый шаблон предоставляет функционал только с текстовыми запросами. Для создания функционала с изображениями необходимо изменить MainActivity.

Для начала настраивается главная активность приложения с использованием Jetpack Compose и устанавливается содержимое активности, определенное в функции App(), с применением определенной темы и стилей, они были в шаблоне.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            EquateTheme {
                Surface(modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background) {
                    App()
                }
            }
        }
    }
}
```

Рисунок 2 – Начальная настройка приложения

Так как в коде используются API, которые находятся в экспериментальном статусе, для их использования требуется явное включение. Чтобы запустить генерацию, приложение использует эффект. Эффект — это функция, которая запускается при определенных условиях. В этом случае эффект запускается, когда переменная generate становится истинной. Эффект вызывает метод generate() в ViewModel. Метод generate() выполняет фактическую генерацию. После выполнения генерации эффект сбрасывает generate в false. Чтобы выбрать изображения из галереи, приложение использует лаунчер. Лаунчер — это функция, которая запускается другой активностью и возвращает результат. В этом случае лаунчер запускает активность выбора нескольких изображений из галереи (рис. 3).

```
@OptIn(ExperimentalComposeUiApi::class)
@Composable
fun App() {

    val context = LocalContext.current
    val keyboardController = LocalSoftwareKeyboardController.current

    val viewModel = remember { MainViewModel() }
    val state by viewModel.state.collectAsStateWithLifecycle()

    var generate by remember { mutableStateOf( value: false) }

    LaunchedEffect(generate) {
        if (generate)
            viewModel.generate()
        generate = false
    }


    val launcher = rememberLauncherForActivityResult(ActivityResultContracts.PickMultipleVisualMedia()) { uris ->
        viewModel.clearData()
        viewModel.addBitmaps(
            uris.map {
                MediaStore.Images.Media.getBitmap(
                    context.contentResolver,
                    it
                )
            }
        )
    }
```

Рисунок 3 – Добавление функции выбора изображения

Далее создаем простой пользовательский интерфейс с заголовком, двумя кнопками и белым фоном. Основной контейнер Box занимает всё доступное пространство на экране и устанавливает белый фон. Колонка Column внутри Box также занимает всё доступное пространство по вертикали и располагает элементы внутри себя по вертикали сверху вниз. Заголовок Text отображает название нейросети "Gemini" с применением стиля MaterialTheme.typography.titleMedium. Он имеет отступы сверху и снизу. Строка Row с кнопками Button имеет отступы со всех сторон. Она выравнивает кнопки по центру по вертикали и распределяет их равномерно по горизонтали. Кнопка "Выбрать изображение" при нажатии запускает лаунчер для выбора изображения из галереи. Кнопка "Очистить" при нажатии вызывает метод viewModel.clearData() для очистки данных в ViewModel (рис. 4).

```
Box(
    modifier = Modifier
        .fillMaxSize()
        .background(Color.White)
        .safeDrawingPadding()
) { this: BoxScope
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Top
    ) { this: ColumnScope
        Text(
            text = "Gemini",
            style = MaterialTheme.typography.titleMedium,
            modifier = Modifier.padding(16.dp)
        )

        Row(
            modifier = Modifier
                .padding(10.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(10.dp),
        ) { this: RowScope
            Button(onClick = {
                launcher.launch(
                    PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageOnly)
                )
            }) { this: RowScope
                Text( text: "Выбрать изображение")
            }

            Button(onClick = {
                viewModel.clearData()
            }) { this: RowScope
                Text( text: "Очистить")
            }

        }
```

Рисунок 4 – Создание интерфейса

Чтобы отобразить выбранные изображения, приложение использует компонент LazyRow. Это контейнер, который предназначен для эффективной прокрутки по горизонтали. Он отображает элементы только тогда, когда они становятся видимыми на экране, что улучшает производительность при большом количестве изображений. В данном случае LazyRow используется

для отображения списка изображений, хранящихся в списке state.bitmapList. Для каждого элемента в списке создается отдельный элемент Image, который отображает битмап изображения. Размер каждого изображения устанавливается в 150 dp. Также добавлено поле ввода, вывода текста и кнопка, выполняющая генерацию (рис. 5).

```
146         LazyRow(
147             contentPadding = PaddingValues(10.dp),
148         ) { this: LazyListScope
149             items(state.bitmapList.size) { this: LazyItemScope index ->
150                 Image(
151                     modifier = Modifier.size(150.dp),
152                     bitmap = state.bitmapList[index].asImageBitmap(),
153                     contentDescription = null
154                 )
155             }
156         }
157
158         Row(
159             modifier = Modifier
160                 .padding(horizontal = 10.dp),
161             verticalAlignment = Alignment.CenterVertically,
162             horizontalArrangement = Arrangement.spacedBy(10.dp),
163         ) { this: RowScope
164
165             OutlinedTextField(
166                 modifier = Modifier
167                     .weight(1f),
168                 value = state.content,
169                 onValueChange = viewModel::onContentChanged,
170                 label = { Text( text: "Ваш вопрос") },
171                 keyboardActions = KeyboardActions(
172                     onDone = { this: KeyboardActionScope
173                         generate = false
174                     }
175                 ),
176                 keyboardOptions = KeyboardOptions(
177                     keyboardType = KeyboardType.Text,
178                     imeAction = ImeAction.Done
179                 )
180             )
```

Рисунок 5 – Создание полей

Следующий код обеспечивает взаимодействие между пользователем и приложением, позволяя запускать процесс генерации и отображать его результаты. Он включает в себя кнопку для запуска генерации, индикатор загрузки для отображения статуса процесса и область для отображения сгенерированного контента (рис. 6).

```
            if (state.isLoading)
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .weight(1f),
                    contentAlignment = Alignment.Center
                ) { this: BoxScope
                    CircularProgressIndicator()
                }
            else
                Column(
                    modifier = Modifier
                        .weight(1f)
                        .verticalScroll(rememberScrollState())
                ) { this: ColumnScope
                    Text(
                        text = state.generatedContent,
                        style = MaterialTheme.typography.titleLarge,
                        modifier = Modifier
                            .padding(16.dp)
                            .fillMaxSize()
                    )
                }
        }

    }

}

@Preview(showSystemUi = true)
@Composable
fun AppPreview() {
    App()
}
```

Рисунок 6 – Создания отображения процесса генерации

Далее создадим класс MainViewModel. Это является важной частью приложения, поскольку он обеспечивает взаимодействие между пользователем и приложением, а также выполняет основную функцию приложения - генерацию контента. Также в коде прописано имя модели, которая будет использоваться для определенных задач. Модель "gemini-pro" используется для обработки текста пользователя, а модель "gemini-pro-vision" для изображений. В данном случае используются две модели.

```
class MainViewModel : ViewModel() {

    data class UIState(
        var bitmapList: List<Bitmap> = listOf(),
        var content: String = "",
        var generatedContent: String = "",
        var isLoading: Boolean = false
    )

    private val textModel by lazy {
        GenerativeModel(
            modelName = "gemini-pro",
            apiKey = BuildConfig.apiKey
        )
    }

    private val visionModel by lazy {
        GenerativeModel(
            modelName = "gemini-pro-vision",
            apiKey = BuildConfig.apiKey
        )
    }

    var state = MutableStateFlow(UIState())
        private set

    fun onContentChanged(text: String) {
        state.update {
            it.copy(
                content = text
            )
        }
    }
```

Рисунок 7 – Выбор модели

Тестируем приложение (рис. 8, 9, 10).

Рисунок 8 – Интерфейс приложения

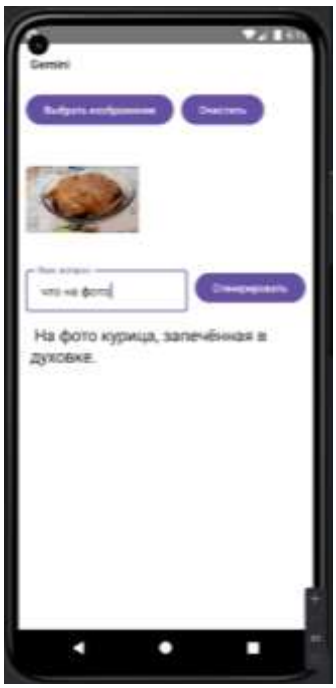

Рисунок 9 – Результат запроса

Рисунок 10 – Результат запроса

Выводы

В данной статье был представлен процесс разработки мобильного приложения с нейросетью Gemini. В результате работы было создано приложение, которое позволяет пользователям отправлять текст и изображения нейросети и получать ответ в виде текста. Приложение было разработано с использованием api, что позволило обеспечить его функциональность, удобство использования и надежность.

Библиографический список

1. Острицова В.А. Применение технологий нейронных сетей при разработке мобильных приложений // Научное обеспечение агропромышленного комплекса. Сборник статей по материалам XI Всероссийской конференции молодых ученых, посвященной 95-летию Кубанского ГАУ и 80-летию со дня образования Краснодарского края. Ответственный за выпуск А. Г. Кощаев. Краснодар, 2017. С. 269-270.
2. Кравченко К.А., Щутский А.С., Креймер А.С. Разработка концепции мобильного приложения для анализа ингредиентов блюд на основе нейронной сети // British Journal of Innovation in Science and Technology. 2018. Т. 3. № 4. С. 39-47.
3. Исаченков А.С., Эльсессер Г.П., Гречкина О.В. Разработка мобильного приложения, предоставляющего пользовательский интерфейс для использования нейронной сети, разработанной с использованием фреймворка Keras. // Известия Тульского государственного университета. Технические науки. 2023. № 9. С. 244-250.
4. Калашникова Д.А. Архитектуры нейронных сетей для мобильных приложений. // Актуальные проблемы авиации и космонавтики. Сборник материалов IX Международной научно-практической конференции, посвященной Дню космонавтики. В 3-х томах. Красноярск, 2023. С. 179-181.
5. Ануфриева Е.Е. Нейронная сеть в мобильных приложениях для поддержки эмоциональной стабильности человека. // Студенческая наука и XXI век. 2022. Т. 19. № 1-1 (22). С. 9-11.
6. URL: https://makersuite.google.com/app/apikey (дата обращения 12.01.2024)