|  |  |  |  |
| --- | --- | --- | --- |
| Systemy rozpoznawania mowy i obrazy | | | |
| Politechnika Świętokrzyska | | | |
| Autorzy |  |  |  |
| Filip Szemraj | Karol Cioć |  | 4ID15A |
| Projekt | Rozpoznawanie gatunku grzybów model wizji i aplikacja | | |

# Model Computer Vision

Fundamentem projektu było opracowanie modelu pozwalającego na klasyfikację gatunku grzyba po przeanalizowaniu jego zdjęcia. Nie wprowadzaliśmy żadnych innych wejściowych, także model bazuje tylko i wyłącznie na przetwarzanym obrazie. Sklasyfikowanych mamy 16 klas:

Indeks 0: Klasa 'Agaricus\_campestris' - Pieczarka polna

Indeks 1: Klasa 'Amanita\_muscaria' - Muchomor czerwony

Indeks 2: Klasa 'Armillaria\_mellea' - Opieńka miodowa

Indeks 3: Klasa 'Boletus\_edulis' - Borowik szlachetny

Indeks 4: Klasa 'Cantharellus\_cibarius' - Kurka (Pieprznik jadalny)

Indeks 5: Klasa 'Imleria\_badia' - Podgrzybek brunatny

Indeks 6: Klasa 'Lactarius\_deliciosus' - Rydz

Indeks 7: Klasa 'Leccinum\_aurantiacum' - Koźlarz czerwony

Indeks 8: Klasa 'Leccinum\_scabrum' - Koźlarz babka

Indeks 9: Klasa 'Macrolepiota\_procera' - Czubajka kania

Indeks 10: Klasa 'Morchella\_esculenta' - Smardz jadalny

Indeks 11: Klasa 'Neoboletus\_erythropus' - Borowik ceglastopory

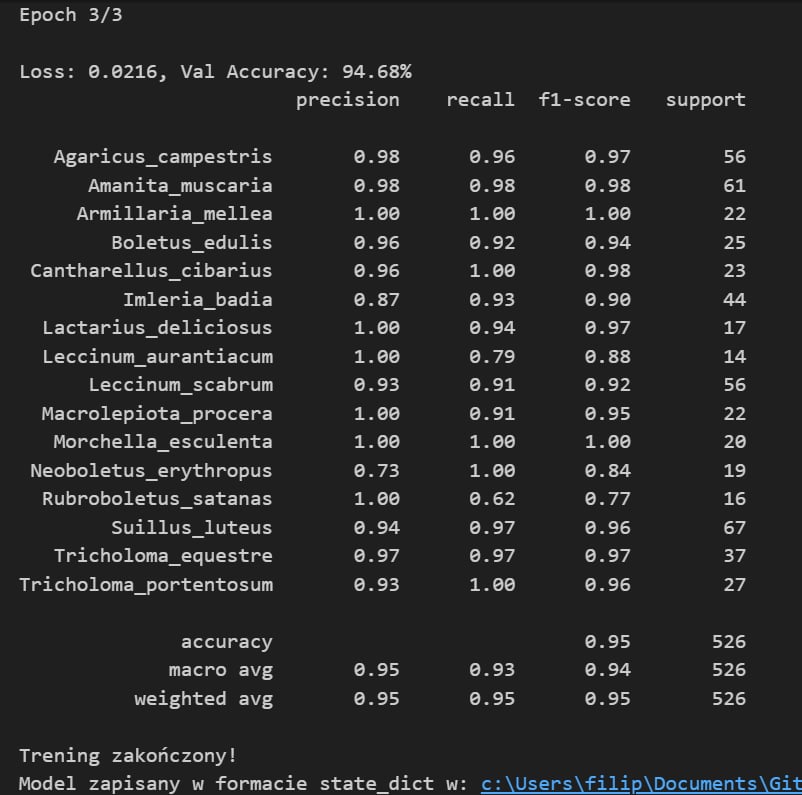
Indeks 12: Klasa 'Rubroboletus\_satanas' - Borowik szatański

Indeks 13: Klasa 'Suillus\_luteus' - Maślak zwyczajny

Indeks 14: Klasa 'Tricholoma\_equestre' - Gąska zielonka

Indeks 15: Klasa 'Tricholoma\_portentosum' - Gąska niekształtna

Na które składa się 4 798 zdjęć dla danych uczących i 526 zdjęć dla danych testowych. Uczenie modelu zajęło około godziny z użyciem karty graficznej NVIDIA GeForce RTX 4060, a to są wyniki:



Wyniki dla wyuczonego modelu.

Proces pozwalający na uzyskanie takich wyników składał się z paru etapów. W trakcie zgłębiania możliwości jakimi dysponujemy do zrealizowania tego projektu dowiedzieliśmy się o modelach wstępnie wyuczonych, takich jak ResNet, EfficientNet, MobileNet czy Vision Transformer, z architektury ostatniego korzystamy. Dodatkowo udało nam się znaleźć projekt ze wstępnie wytrenowanymi modelami idealnie pasujący do naszego tematu, czyli – „***Danish Fungi 2020 - Not Just Another Image Recognition Dataset***”[[[1]](#footnote-1)]. Projekt ten bazuje na 110GB obrazów skutkujących 1 604 klasami, w skład których wchodzi 266 344 obrazy uczące i 29 594 testowe. Udostępnione wyniki dla wyuczonych modeli wskazywały, że najlepsze rezultaty można osiągnąć z ViT[[[2]](#footnote-2)]. Na początku chcieliśmy użyć największego zestawu wstępnie wyuczonych wag ViT-Large/16 bazującego na obrazach o wymiarach 224x224 o rozmiarze 1.1GB, jednak okazało się, że nie dysponujemy wystarczającymi zasobami w postaci pamięci karty graficznej, aby dopasować te wagi do naszych klas. Z tego powodu przeszliśmy na ViT-Base/16 bazującym na 384x384 rozdzielczości obrazów, jednak ważącym już tylko 333MB, jest to drugi w kolejności model jeśli chodzi o klasyfikację obrazów podaną we wcześniej wspomnianym rankingu. Oprócz samej klasyfikacji dostępnej w projekcie sprawdzaliśmy także szereg modeli w kontekście uczenia się na naszych danych, lecz żaden z nich nie dawał nam aż tak dobrych rezultatów, a były to między innymi: EfficientNet-B0 o rozdzielczości 224x224 jak i również 384x384, MobileNet-V2 rozdzielczość 299x299.

Wśród wymienionych modeli występują dwie architektury, czyli CNN (ang. Convolutional Neural Network) i ViT (ang. Vision Transformer). MobileNet i EfficientNet korzystają z CNN.

* CNN

Convolutional Neural Networks to sieci neuronowe zaprojektowane do przetwarzania danych wizualnych. Korzystają z warstw splotowych, które analizują obrazy w sposób lokalny i hierarchiczny – od prostych krawędzi do złożonych wzorców. CNN są efektywne i wydajne, szczególnie małych zbiorach danych, gdzie większe znaczenie mają lokalne różnice.

* ViT

Vision Transformer to architektura oparta na mechanizmie samoatencji (ang. self-attention), znanym z przetwarzania języka naturalnego w modelach takich jak BERT czy GPT. ViT dzieli obraz na fragmenty (patchy) i analizuje ich globalne zależności, dzięki czemu świetnie nadaje się do dużych zbiorów danych. Jest to nowsze podejście w wizji komputerowej, które różni się od lokalnej analizy stosowanej w CNN.

Nasz zbiór danych pozyskaliśmy głównie ze strony <https://mushroomobserver.org/>, korzystając z proponowanego przez autorów API, którego opis można znaleźć tutaj <https://github.com/MushroomObserver/mushroom-observer/blob/main/README_API.md>. Posłużył nam do tego prosty skrypt Python, który buduje ścieżki do zdjęć i pobiera je z odstępem czasowym równym 6 sekund ze względu na zabezpieczenia przeciążenia strony. Do budowania ścieżek potrzebne były ID zdjęć, które zwraca API, oto jedno z zapytań dotyczące podgrzybka brunatnego: <https://mushroomobserver.org/api2/images?name=Imleria%20badia&detail=none&content_type=jpg&page=1&ok_for_export=true>

Dla niektórych klas obrazów było za mało, z czym poradziliśmy sobie poprzez ekstrakcje zdjęć (ang. Scraping) z Google, uwzględniając jednak licencje pobieranych zdjęć.

Do wygenerowania modelu stworzyliśmy skrypt Python, który ładuje i przygotowuje zdjęcia, trenuje model walidując go i na koniec zapisuje. Dodatkowo oddzielnie zrealizowaliśmy konwersje modelu do formatu onnx. Ze względu na złożoność skrypt jest dołączony jako załącznik. Wraz z kodem zawarte są komentarze objaśniające jego działanie. Wartym dodania jest fakt, że ze względu na architekturę systemu operacyjnego Windows nie było możliwe dodanie większej ilości wątków dla DataLoader, ze względu na sposób w jaki Windows zarządza procesami i wątkami w Python. Dane uczące i testowe, w kodzie są obsługiwane w postaci 16 elementowych grup co pozwala na zmniejszenie wymagań pamięciowych i równoległe (nie w przypadku systemu Windows) przetwarzanie grup danych tym samym przyśpieszając obliczenia i stabilizując samo uczenie. Grupy próbek mają zredukowany szum i są bardziej wyrównane niż pojedyncze zdjęcia. Do uczenia używana jest krzyżowa strata entropii, którą można opisać za pomocą wzoru:

Opis dostępny pod <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

# Aplikacja mobilna

Do przedstawienia działania modelu stworzyliśmy aplikację mobilną z wykorzystaniem technologii React Native. Umożliwia ona wczytanie zdjęcia grzyba z galerii urządzenia w celu jego rozpoznania. Po wybraniu zdjęcia użytkownik zostaje przekierowany do ekranu, na którym jest wyświetlany wynik w postaci nazwy łacińskiej i polskiej rozpoznanego grzyba.

## Opis kodu

Kopiowanie modelu ONNX z zasobów aplikacji do wewnętrznej pamięci urządzenia, ponieważ biblioteka onnxruntime wymaga dostępu do pliku modelu z systemu plików.

const copyModelToInternalStorage = async (): Promise<string> => {

    const sourcePath = "model.onnx";

    const destinationPath = `${RNFS.DocumentDirectoryPath}/model.onnx`;

    await RNFS.copyFileAssets(sourcePath, destinationPath);

    return destinationPath;

  };

Załadowanie modelu ONNX, przetwarzanie obrazu wejściowego, uruchomienie modelu aby uzyskać wynik.

const runModel = async (imageUri: string) => {

    let session: InferenceSession | null = null;

    try {

      setLoading(true);

      const modelPath = await copyModelToInternalStorage();

      session = await InferenceSession.create(modelPath);

      const inputTensor = await preprocessImage(imageUri);

      const feeds = { input: inputTensor };

      const output = await session.run(feeds);

      const outputTensor = output["output"];

      const prediction = interpretOutput(outputTensor);

      setResult(prediction);

    } catch (error) {

      console.error("Błąd podczas uruchamiania modelu ONNX:", error);

    } finally {

      setLoading(false);

      if (session) {

        session.release();

      }

    }

  };

Przetwarzanie obrazu na Tensor, który może być użyty jako wejście do modelu.

const preprocessImage = async (imageUri: string): Promise<Tensor> => {

    try {

      const resizedImage = await ImageResizer.createResizedImage(

        imageUri,

        384,

        384,

        "JPEG",

        100,

        0,

        undefined,

        false

      );

      return new Tensor(

        "float32",

        await normalizeImage(resizedImage.uri),

        [1, 3, 384, 384]

      );

    } catch (error) {

      if (error instanceof Error) {

        throw new Error(`Błąd podczas przetwarzania obrazu: ${error.message}`);

      } else {

        throw new Error("Błąd podczas przetwarzania obrazu: Nieznany błąd");

      }

    }

  };

Normalizacja obrazu do formatu odpowiedniego dla modelu ONNX.

Normalizuje wartości pikseli dla każdego kanału RGB według wzoru:

const normalizeImage = async (uri: string): Promise<Float32Array> => {

    try {

      const imageBuffer = await RNFS.readFile(uri, "base64").then((base64) =>

        Buffer.from(base64, "base64")

      );

      const { data, width, height } = decodeImage(imageBuffer);

      const normalized = new Float32Array(3 \* 384 \* 384);

      const mean = [0.485, 0.456, 0.406];

      const std = [0.229, 0.224, 0.225];

      for (let i = 0; i < height; i++) {

        for (let j = 0; j < width; j++) {

          const pixelIndex = (i \* width + j) \* 4;

          const r = data[pixelIndex] / 255.0;

          const g = data[pixelIndex + 1] / 255.0;

          const b = data[pixelIndex + 2] / 255.0;

          normalized[i \* 384 + j] = (r - mean[0]) / std[0];

          normalized[384 \* 384 + i \* 384 + j] = (g - mean[1]) / std[1];

          normalized[2 \* 384 \* 384 + i \* 384 + j] = (b - mean[2]) / std[2];

        }

      }

      return normalized;

    } catch (error) {

      if (error instanceof Error) {

        throw new Error(`Błąd podczas normalizacji obrazu: ${error.message}`);

      } else {

        throw new Error("Błąd podczas normalizacji obrazu: Nieznany błąd");

      }

    }

  };

## Przedstawienie rozwiązania

|  |  |  |
| --- | --- | --- |
| C:\Users\Karolek\Desktop\home.png | C:\Users\Karolek\Desktop\imagePicker.png | C:\Users\Karolek\Desktop\result.png |

1. [] Lukáš Picek, Milan Šulc, Jiří Matas, Thomas S. Jeppesen, Jacob Heilmann-Clausen, Thomas Læssøe, Tobias Frøslev; Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2022, pp. 1525-1535 - https://sites.google.com/view/danish-fungi-dataset [↑](#footnote-ref-1)
2. [] https://paperswithcode.com/sota/image-classification-on-df20 [↑](#footnote-ref-2)