Projekt 1

Detektor Emocji

Tematem mojego pierwszego projektu z Inteligencji Obliczeniowej jest aplikacja ***Detektor Emocji "MoodSense"***. Podczas jego tworzenia wyszkoliłam kilka różnych modeli, wykorzystałam dwie różne bazy danych, oraz przedstawiłam praktyczne zastosowanie wykrywania i reagowania na poszczególne emocje. Opracowanie zawiera szczegółowy opis mojego programu oraz treningu sieci neuronowej.

---

1. Model sieci neuronowej

Bazy danych i preprocessing

Do szkolenia swoich modeli wykorzystałam dwie bazy danych - **The Extended Cohn-Kanade (CK+)**, posiadającą około 920 zdjęć emocji, i **FER-2013** z ponad 28 tysiącami zdjęć. Pracuję na siedmiu emocjach - *złość*, *niesmak*, *strach*, *smutek*, *zaskoczenie*, *szczęście*, *neutralność/pogarda*. CK+ posiada dosyć proste, nie zróżnicowane dane, a FER-2013 bardzo różnorodne, nie zawsze od przodu i w dużej liczbie.

Podczas preprocessingu tworzę zbiory treningowe i testowe z każdej klasy, ustalam rozmiar ich partii i wymiary zdjęć, w obecnym kodzie są to wartości 16 i 48x48. Następnie przeprowadzam prosty one-hot encoding, aby przedstawić kategoryczne klasy jako dane numeryczne.

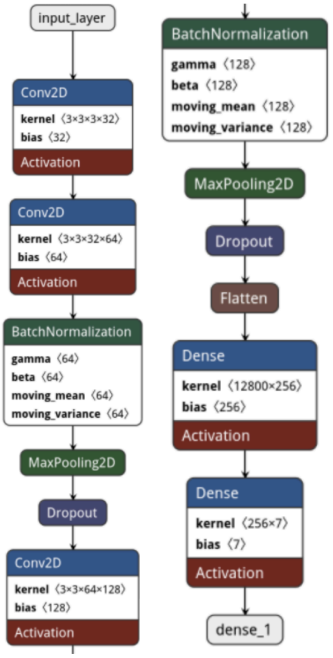
Używałam również funkcji `prepare_data`, która robiła downsampling bazy danych. Jej zadaniem było wyrównanie wielkości każdej klasy, aby miały tyle samo zdjęć. Jednak modele osiągały lepsze wyniki, gdy liczba danych pozostawała w oryginalnym stanie, dlatego zakomentowałam tę funkcję. W przypadku datasetu FER-2013, ponieważ jest on wystarczająco duży, ręcznie usunęłam katalog `/test` i wyciągnęłam katalogi klas z `/train` do głównego katalogu datasetu.

Klasyfikacja i optymalizacja

Do szkolenia detektora emocji wykorzystałam deep learning, a dokładniej konwolucyjne sieci neuronowe. Idealnie nadają się do analizy danych wielowymiarowych, m.in. zdjęć i dźwięku. Z każdą warstwą, CNN zwiększa swoją dokładność i coraz lepiej rozpoznaje obraz, dzięki czemu ostatecznie jest w stanie wykryć najistotniejsze cechy. Stworzyłam bardzo prosty model, składający się z jedynie

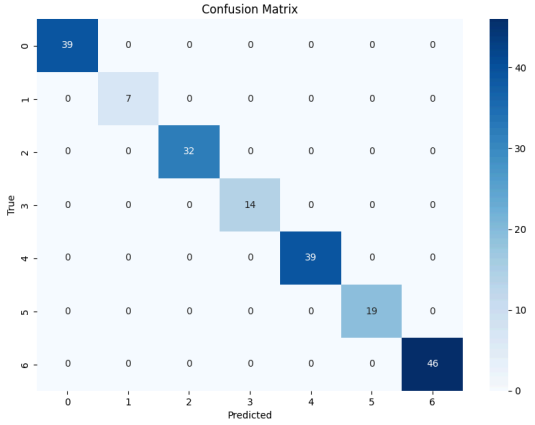
trzech warstw konwolucyjnych. Model zawiera też warstwy poolingu do zmniejszania rozmiaru wyników, normalizacji wsadowej do przyspieszenia uczenia, a także warstwę spłaszczającą, zmieniającą kształt wyniku na jednowymiarowy. Warstwy Dropout losowo wyłączają część neuronów podczas treningu, by zapobiegać przeuczeniu. Zastosowałam również Early Stopping, który zapobiega zbyt długim okresom bez poprawy i w razie potrzeby przywraca lepsze wyniki.

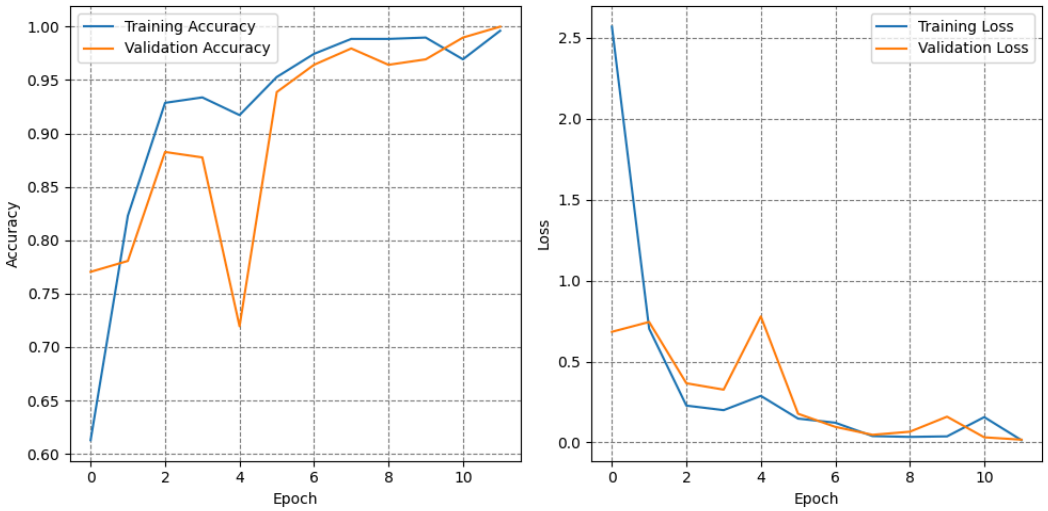
```
model = Sequential([
  Input(shape=(48, 48, 3)),
  Conv2D(32, (3, 3), activation="relu", kernel_initializer="he_uniform", padding="same"),
  Conv2D(64, (3, 3), activation="relu"),
  BatchNormalization(),
  MaxPooling2D(pool_size=(2, 2)),
  Dropout(0.25),
  Conv2D(128, (3, 3), activation="relu"),
  BatchNormalization(),
  MaxPooling2D(pool_size=(2, 2)),
  Dropout(0.25),
  Flatten(),
  Dense(256, activation="relu", kernel_initializer="he_uniform"),
  Dense(7, activation="sigmoid")
])

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```



Podczas pracy nad projektem wytrenowałam łącznie pięć modeli - trzy na bazie CK+ i dwa na FER-2013. Swoje obserwacje skupiłam na trzecim i czwartym modelu.

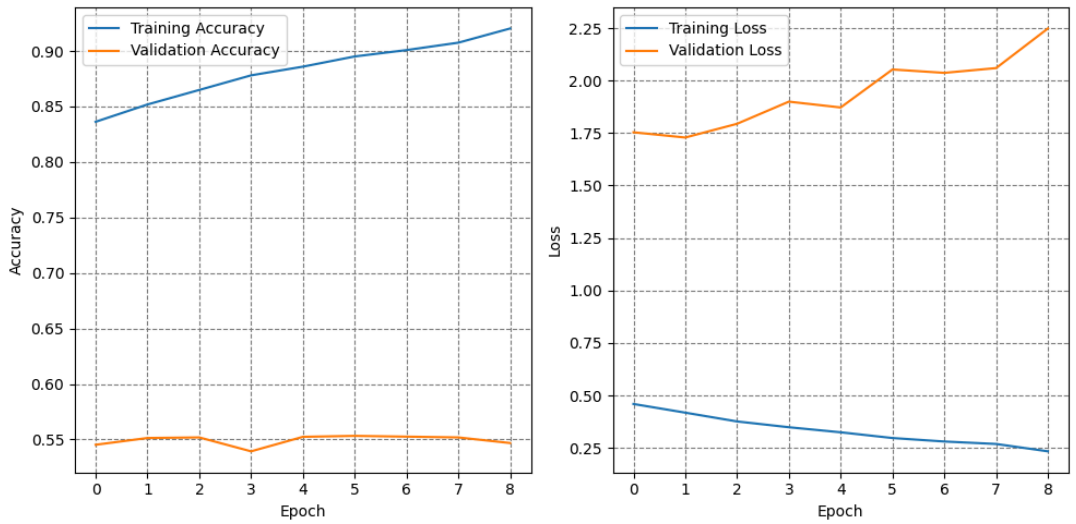
Dataset CK+ jest mały i niezróżnicowany. Zdjęcia są do siebie podobne i łatwo je rozpoznać. Pierwsze dwa modele wytrenowałam na dodatkowo zmniejszonej bazie przez downsampling. Jeden miał architekturę z poradnika, drugi napisaną przeze mnie i maksymalnie udało mi się wyszkolić je na 94% i 96%, przy czym wykresy obu wskazują na good fit. Wyniki w postaci macierzy błędu i krzywych uczenia się można zobaczyć w folderze /plots. Trzeci model uczył się na oryginalnej bazie i już w 12 epoce osiągnął 100% dokładności.





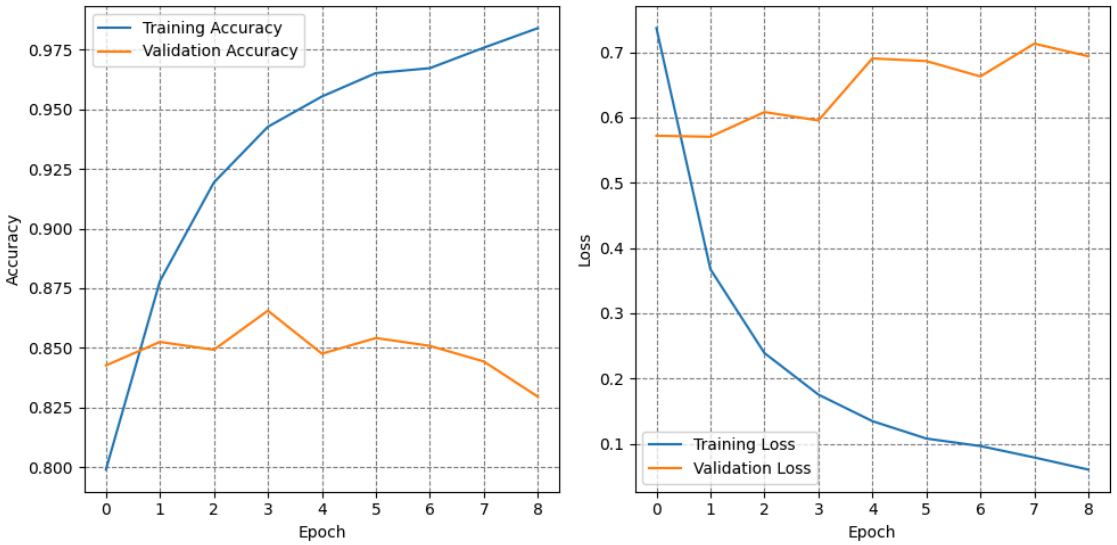
Na heatmapie widać, że model jest precyzyjny, a na wykresach krzywych uczenia się, że jest dobrze wytrenowany. Przy dokładności obie krzywe rosną do pewnego miejsca, a przy stracie obie maleją. Dodatkowo model uczył się bardzo szybko, ponieważ trwało to mniej niż 10 minut.

Dataset FER-2013 jest duży (oryginalnie zawiera ponad 35 tysięcy obrazów) i bardzo zróżnicowany. Wpłynęło to zdecydowanie negatywnie na czas obliczeń i precyzyjność. Główny model potrzebował około 20 minut na jedną epokę i miał wysoką stratę walidacyjną. W łącznie 20 epokach okazywał rzadkie poprawy dokładności i przy 55% krzywe uczenia wskazywały na to, że model nie był w stanie nauczyć się zbioru treningowego.



Może to być spowodowane prostotą modelu, jednak na bardziej złożonej architekturze, z większą ilością warstw konwolucyjnych, przedstawiał bardzo podobne wyniki jak na zdjęciu powyżej.

Kiedy dotrenowałam ten model na zmniejszonym datasecie FER-2013, na którym zrobiłam downsampling do równej liczby zdjęć w każdej klasie (zostało ich wtedy łącznie 3052), model osiągnął 85.56% dokładności po 9 epokach i wyszkolił się szybko. Mimo to, nadal nie był dobrze dopasowany, a wręcz wskazywał na przeuczenie. Krzywa dokładności walidacyjnej malała, kiedy treningowej rosła, i odwrotnie w krzywej straty. Model bardziej "zapamiętywał" dane, zamiast uczyć się z nich.



Zaciekawiona tym nagłym skokiem dokładności, postanowiłam również sprawdzić model wytrenowany od podstaw na zmniejszonym datasecie. Niestety, w tym przypadku nie poradził sobie. Z 20 epok czternasta była ostatnią, w której nastąpił wzrost, a najwyższą osiągniętą dokładnością było 35.41%. Krzywe uczenia się wskazywały na przeuczenie modelu.

Ponieważ modele uczone na FER-2013 nie przejawiały oczekiwanych rezultatów, do głównej aplikacji wykorzystałam trzeci model wytrenowany na CK+. Jest on najbardziej precyzyjny i szybko się wyszkolił, dzięki czemu wydał mi się najbardziej optymalny do rozpoznawania emocji. Przy samym wykrywaniu emocji w czasie rzeczywistym również sprawdza się bardzo dobrze.

---

2. Funkcjonalność aplikacji

Instrukcja użycia programu (po angielsku) znajduje się w pliku README.md.

Detektor twarzy i emocji

Kaskada Haara to algorytm detekcji obiektów, który wykorzystuje cechy lokalne obrazu bazujące na zmianach intensywności pikseli. Ta technika jest często stosowana do szybkiej detekcji obiektów w czasie rzeczywistym, dlatego najlepiej pasowała do mojego projektu. Klasyfikator kaskady Haara jest wbudowany w bibliotekę OpenCV. Użyłam do niego wytrenowany klasyfikator, którego struktura została zawarta w pliku haarcascade_frontalface_default.xml.

W tworzeniu detektora korzystałam z pomocy projektu użytkownika Atulapra na GitHubie, modyfikując go według potrzeb mojego modelu. Na przykład, image_dataset_from_directory sprawia, że czarno-białe obrazy są rozpoznawane przez model jako trzykanałowe, więc przy przewidywaniu emocji wymagane było przywrócenie koloru. Dodałam także dodatkowe rozszerzenia wymiarów tablicy obrazu, aby projekt działał.

Detektor włącza kamerkę internetową, a następnie znajduje twarz na obrazie, oznacza ją różową ramką i podpisuje wykrytą emocją. Działanie aplikacji pokazuję na modelu wytrenowanym na datasecie CK+ z dokładnością 100%. Mimo tego, że dane były raczej proste i w małych ilościach, detektor wykrywa większość emocji poprawnie, myląc ze sobą głównie te podobne do siebie. Strach zazwyczaj przedstawia jako zaskoczenie oraz ma problemy ze złością, niesmakiem i neutralnością/pogardą. Jest to najprawdopodobniej wynikiem braku różnorodności danych i tego, że te emocje są do siebie bardzo podobne, więc rozróżnienie ich od siebie bywa ciężkie.

```python
def run_detector():
    model = load_model("detection_model_3.keras")
    cv2.ocl.setUseOpenCL(False)
    labels = { 0: "anger", 1: "neutral", 2: "disgust", 3: "fear", 4: "happiness", 5: "sadness", 6: "surprise" }

    webcam = cv2.VideoCapture(0)
    print("||   Started the detection program!")
    print("||   Press Q when you want to exit.")

    detected_emotions = []
    last_action_time = time.time()

    while True:
        ret, frame = webcam.read()
        if not ret:
            break

        facecasc = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = facecasc.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (225, 79, 215), 2)
            roi_gray = gray[y:y + h, x:x + w]
            colored_image = crop_and_color(roi_gray)
            prediction = model.predict(colored_image, verbose=0)
            maxindex = int(np.argmax(prediction))

            cv2.putText(frame, labels[maxindex], (x+20, y-60), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
            detected_emotions.append(labels[maxindex])

        if time.time() - last_action_time > 30:
            most_common_emotion = max(set(detected_emotions), key=detected_emotions.count)
            perform_action_for_emotion(most_common_emotion)
            detected_emotions = []
            last_action_time = time.time()

        cv2.imshow("MoodSense", cv2.resize(frame, (600, 400), interpolation=cv2.INTER_CUBIC))

        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    webcam.release()
    cv2.destroyAllWindows()
    print("||   The app is closing. Goodbye!")
```

Funkcjonalność aplikacji

Program można uruchomić na trzy sposoby:

1) Bez flagi - wyjaśni, do czego służy program oraz pokaże flagi z ich działaniem.
2) Z flagą -t lub --train - przygotuje dane, wytrenuje model, przedstawi jego macierz błędu i krzywe uczenia się.
3) Z flagą -r lub --run - uruchomi główną aplikację wykrywającą i reagującą na emocje. Można z niej wyjść, przyciskając Q na klawiaturze.

Oprócz samego wykrywania emocji, program również reaguje na nie. Dla ułatwienia, ponieważ jedna emocja nie zgadzała się pomiędzy dwiema bazami danych, a jest całkiem podobna, neutralność/pogardę podpisałam jako zwykłą neutralność.

Przez 30 sekund detektor zbiera dane na temat wykrytych emocji, po czym zlicza je i szuka tej, która wystąpiła najczęściej, by na nią zareagować. W przypadku negatywnych emocji, takich jak złość, smutek lub niesmak, program puszcza relaksującą muzykę lub pokazuje zdjęcie uroczego zwierzątka. Cieszy się z dobrego humoru i życzy miłego dnia, gdy użytkownik jest szczęśliwy, a gdy neutralny - mówi, że nie będzie przeszkadzał. Za to gdy widzi zaskoczenie, proponuje popcorn. Relaksująca muzyka pochodzi z biblioteki darmowych utworów Pixabay.

Podsumowując, nauczanie modelu na minimalistycznej architekturze, ale różnych bazach danych wraz ze zmianami w nich, pokazało interesujące rezultaty. Gorsze wyniki wychodziły w przypadku zmniejszonych danych, często w połączeniu z przeuczeniem, jednak w przypadku liczby obrazów bliskiej 30 tysięcy model nie był w stanie nauczyć się datasetu.

Model o dokładności 100%, mimo treningu na prostych, podobnych do siebie danych, w połączeniu z klasyfikatorem kaskady Haara bardzo dobrze i niemal w pełni precyzyjnie wykrywa emocje w czasie rzeczywistym.

Detektor emocji jest zdecydowanie popularnym projektem, który jednak może być przedstawiony na wiele sposobów. Reakcja aplikacji na emocje nadaje personalny charakter i może czynić projekt bardziej interesującym, poprawiającym humor użytkownikom.

Bibliografia:

Kaggle - FER-2013: https://www.kaggle.com/datasets/msambare/fer2013
Kaggle - CK+: https://www.kaggle.com/datasets/davilsena/ckdataset
Tutorial from Skillcate AI: https://medium.com/@skillcate/emotion-detection-model-using-cnn-a-complete-guide-831db1421fae
GitHub project from user Atulapra: https://github.com/atulapra/Emotion-detection
Netron model visualization: https://netron.app
Pixabay free music: https://pixabay.com/
OpenCV documentation/tutorials: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
OpenCV Cascade Classifier: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
Haar Cascade Classifier: https://machinelearningmastery.com/using-haar-cascade-for-object-detection/
Deep Learning: https://www.datacamp.com/tutorial/introduction-to-deep-neural-networks
CNN: https://www.ibm.com/topics/convolutional-neural-networks
Learning curves: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/
Additional help: ChatGPT, GitHub Copilot, presentations from lectures and tasks/tutorials from laboratories