Politechnika Wrocławska

Sprawozdanie

Sieci neuronowe

Wykonał: Julia Rojek (272529), Emila Kowalczyk(272539)

Semestr zimowy: 2024/25

Przedmiot: Sieci Neuronowe

Data zajęć: Czwartek

Godzina zajęć: 17.05-18.45

Spis treści

Spis rysunków

Spis tabel

1. Opis projektu

Głównym celem projektu było zaprojektowanie i zaimplementowanie konwolucyjnej sieci neuronowej (CNN) służącej do rozwiązania problemu klasyfikacji obrazów. W ramach projektu skupiono się na rozpoznawaniu dwóch emocji smutku (SAD) i radości (HAPPY).

2. Przygotowanie Danych

2.1 Opis danych

W ramach projektu wykorzystano zestaw danych pochodzący z platformy Kaggle, który w oryginalnej wersji zawierał obrazy reprezentujące 7 różnych emocji. Na potrzeby implementacji zdecydowano się ostatecznie na wykorzystanie dwóch podstawowych stanów emocjonalnych: szczęścia (HAPPY) i smutku (SAD), gdyż przy testowaniu pełnego zestawu danych wyniki okazały się dość słabe. Natomiast wykorzystanie tylko dwóch emocji umożliwiło otrzymanie najbardziej optymalnego wyniku.

Zbiór danych składa się ze zdjęć przedstawiających ludzkie twarze, przy czym obrazy charakteryzują się znaczącym zróżnicowaniem pod względem ujęcia. Twarze nie są wycentrowane w kadrze i pojawiają się pod różnymi kątami nachylenia. W niektórych przypadkach występują częściowe zasłonięcia obszaru twarzy, na przykład przez rękę, co dodatkowo zwiększa złożoność zadania rozpoznawania.

Rysunek 1 Przykładowe zdjęcia przedstawiające klasę „HAPPY”

Rysunek 2 Przykładowe zdjęcia przedstawiające klasę „SAD”

Podczas analizy zbioru zdjęć zaobserwowano nieścisłość w reprezentowanych klasach. W folderach przypisanych do konkretnych emocji sporadycznie pojawiają się zdjęcia reprezentujące inne stany emocjonalne. Przykładowo, w katalogu zawierającym obrazy szczęśliwych twarzy (HAPPY) można znaleźć zdjęcia wyrażające złość, zaskoczenie czy neutralny wyraz twarzy. Podobne zjawisko zachodzi w katalogu obrazującym emocję „SAD", znajdują się tam przykłady emocji, które reprezentują radość, zaskoczenie. Choć przypadki te są stosunkowo rzadkie, wpływają one jednak na skuteczność uczenia sieci i jej końcową wydajność.

Rysunek 3 Przykładowe obrazy, na których twarze są częściowo przesłonięte

Rysunek 4 Przykład zdjęć, przedstawiające inna emocje niż „sad", znajdujące się w taj klasie

2.2 Charakterystyka Danych Wejściowych

Zbiór danych trenningowycg składa się z 3021 obrazów, w tym 1774 zdjęć reprezentujących klasę "happy" oraz 1247 zdjęć reprezentujących klasę "sad". Wszystkie obrazy w zbiorze mają jednolity format JPG oraz stałe wymiary 48x48 pikseli. Są to obrazy kolorowe, zapisane w przestrzeni RGB (3 kanały kolorów). Analiza wartości pikseli wykazała, że obrazy wykorzystują pełny zakres tonalny charakterystyczny dla 8-bitowej głębi kolorów, od 0 (czerń) do 255 (biel). Badanie średnich wartości pikseli w obrazach ujawniło znaczące zróżnicowanie w jasności poszczególnych zdjęć. Najciemniejsze obrazy charakteryzują się średnią wartością pikseli około 49-50, podczas gdy najjaśniejsze osiągają średnie wartości w zakresie 180-186. Ta różnorodność w jasności obrazów może mieć istotny wpływ na proces uczenia sieci neuronowej.

2.3 Preprocessing Danych

Pierwszym krokiem było przekształcenie obrazów z przestrzeni kolorów RGB (3 kanały) do skali szarości (1 kanał). Transformacja została zrealizowana przy użyciu funkcji load_img z parametrem color_mode='grayscale', co pozwoliło na redukcję złożoności danych przy zachowaniu istotnych cech obrazów. Następnie przeprowadzono normalizację wartości pikseli, przekształcając je z oryginalnego zakresu [0,255] do przedziału [0,1]. Operacja ta została wykonana poprzez

konwersję typu danych na float32 i podzielenie wszystkich wartości przez 255.0. Użycie normalizacja jest kluczowe dla stabilnego procesu uczenia sieci neuronowej. W kolejnym etapie preprocessingu przygotowano etykiety klas. Zastosowano kodowanie one-hot encoding, wykorzystując funkcję tf.keras.utils.to_categorical. Przyjęto mapowanie, gdzie klasa "happy" została oznaczona jako 0, a klasa "sad" jako 1. Przez cały proces preprocessingu zachowano jednolity rozmiar obrazów wynoszący 48x48 piksel

2.4 Podział danych

Zbiór zdjęć został podzielony na części treningową i testową. Liczebność poszczególnych klas wygląda następująco:

Folder testowy zawiera łącznie 12045 obrazów

- 7215 zdjęć przedstawiających klasę ”happy”
- 4830 zdjęć przedstawiających klasę ”sad”

Jednak przyjęto pewne ograniczenia. Dla każdej klasy wczytywane jest tylko 950 zdjęć.

W przypadku zbioru treningowego (train) wprowadzono limit 3000 zdjęć dla każdej klasy. Dane treningowe zostały następnie podzielone na dwie części:

- 80% danych to właściwy zbiór treningowy, dzięki niemu uczymy model
- 20% danych to zbiór walidacyjny, pozwala określić bieżącą jakość modelu

Realizacja podziału zbioru treningowego i walidacyjnego została zrealizowana dzięki funkcji train_test_split z biblioteki scikit-learn, z parametrami test_size=0.2 oraz random_state=42. Zastosowanie stałego ziarna losowości (random_state) zapewnia powtarzalność eksperymentów i możliwość porównywania różnych wariantów modelu w spójnych warunkach.

3. Architektura i Implementacja Sieci

3.1. Struktura Sieci

Sieć składa się z warstwy wejściowej typu input, 4 warstw konwulucyjnych (Convolutional Layers), 3 warstw gęstych (Fully Connected Layers) oraz warstwy wyjściowej typu dens funkcją aktywacji softmax, która umożliwia przypisanie obrazu do jednej z dwóch klas emocji.

Sieć zaczyna się od warstwy wejściowej, która przyjmuje obrazy o rozdzielczości 48x48 pikseli w odcieniach szarości. Pierwsza część modelu to cztery warstwy konwolucyjne, które mają na celu wyodrębnienie istotnych cech z obrazu. Każda z tych warstw jest następnie uzupełniona o operację MaxPooling, aby zredukować rozmiar przestrzenny obrazów i wyizolować najbardziej znaczące cechy. W każdej z warstw konwolucyjnych zastosowano również BatchNormalization i Dropout, które pomagają w stabilizacji procesu uczenia oraz zapobiegają przeuczeniu modelu.

Po warstwach konwolucyjnych, dane są spłaszczane i przechodzą przez trzy warstwy gęste (fully connected), które są odpowiedzialne za ostateczną klasyfikację. Każda z tych warstw zawiera regularizację L2 oraz funkcję aktywacji ReLU, a także Dropout w celu dalszego zapobiegania przeuczeniu. Na końcu sieć kończy się warstwą wyjściową z 2 neuronami, odpowiadającymi za

klasyfikację do dwóch klas emocji. Funkcja aktywacji softmax zapewnia probabilistyczne wyjście, wskazując, do której klasy dany obraz należy.

3.2. Parametry Uczenia

Do wytrenowania sieci, aby uzyskała jak najlepszy wynik (accuracy) zostały zmodyfikowane i ustawione na nastepujące parametry:

Podstawowe parametry uczenia :

- Learning rate (współczynnik uczenia): 0.000002
- Liczba epok : 100
- Batch size : 64
- Weight decay: 0.004
- Optymalizator: AdamW z learning_rate=0.0005

Parametry regularyzacji:

- Dropout: o wartościach 0.3 dla warstw konwolucyjnych i 0.4-0.5 dla warstw gęstych
- L2 regularization: zastosowana w warstwach konwolucyjnych i gęstych
- Batch Normalization: po każdej warstwie konwolucyjnej i gęstej

Mechanizmy zapobiegające przetrenowaniu:

- Early Stopping z patience=8 i min_delta=0.001
- ReduceLROnPlateau z factor=0.1, patience=4 i min_lr=1e-6
- ModelCheckpoint zapisujący najlepszy model według accuracy

Parametry warstw:

- Warstwy konwolucyjne: 32, 64, 128, 256 filtrów
- Warstwy gęste: 150, 120, 70 neuronów
- Funkcja aktywacji: ReLU dla warstw ukrytych, softmax dla warstwy wyjściowej

Dzięki zastosowaniu zoptymalizowanych parametrów dla danego zbioru zdjęć testowych uzyskano optymalne warunki uczenia oraz zapobiegło to przeuczeniu, które było częstym zjawiskiem podczas prób dobierania odpowiednich wartości.

Typ warstwy	***Parametry***	***Funkcja aktywacji***	***Regularyzacja***
Input	(48, 48, 1)	-	-
Conv2D	32 filtrów, (3,3)	ReLU	L2, BatchNorm, Dropout(0.3)
MaxPooling2D	(2,2)	-	-
Conv2D	64 filtrów, (3,3)	ReLU	L2, BatchNorm, Dropout(0.3)
MaxPooling2D	(2,2)	-	-

Conv2D	128 filtrów, (3,3)	ReLU	L2, BatchNorm, Dropout(0.3)
MaxPooling2D	(2,2)	-	-
Conv2D	256 filtrów, (3,3)	ReLU	L2, BatchNorm, Dropout(0.3)
MaxPooling2D	(2,2)	-	-
Flatten	-	-	-
Dense	150 neuronów	ReLU	L2, BatchNorm, Dropout(0.4)
Dense	120 neoronów	ReLU	L2, BatchNorm, Dropout(0.4)
Dense	70 neuronów	ReLU	L2, BatchNorm, Dropout(0.5)
Dense (output)	NUM_CLASSES	Softmax	-

Tabela 1 Tabele pokazująca architekturę sieci neuronowej

4. Proces Uczenia i Eksperymenty

4.1 Mechanizm Uczenia

W zastosowanym modelu sieci neuronowej wykorzystano zaawansowany mechanizm uczenia. Głównym elementem jest optymalizator ***AdamW*** (adaptive moment estimation with weight decay), który operuje ze współczynnikiem uczenia (learning rate) 0.000002 oraz parametrem weight decay ustawionym na 0.004. Jako funkcję straty przyjęto categorical crossentropy, która jest szczególnie efektywna w zadaniach klasyfikacji wieloklasowej.

Proces uczenia jest optymalizowany przez kilka mechanizmów kontrolnych. ReduceLROnPlateau monitoruje wartość funkcji straty na zbiorze walidacyjnym i w przypadku braku poprawy przez 4 epoki redukuje współczynnik uczenia o rząd wielkości (factor=0.1), nie pozwalając mu jednak spaść poniżej 1e-6. Równolegle działa mechanizm Early Stopping, który zatrzymuje proces uczenia, jeśli przez 8 epok nie nastąpi poprawa większa niż 0.001, przywracając jednocześnie najlepsze osiągnięte wagi.

Model wykorzystuje również zaawansowane techniki regularyzacji. Zastosowano regularyzację L2 na warstwach konwolucyjnych i gęstych, warstwy Dropout o współczynnikach od 0.3 do 0.5 oraz normalizację wsadową (Batch Normalization), co wspólnie zapobiega przetrenowaniu i stabilizuje proces uczenia. Dodatkowo, najlepszy model jest automatycznie zapisywany dzięki mechanizmowi ModelCheckpoint, który monitoruje dokładność na zbiorze walidacyjnym.

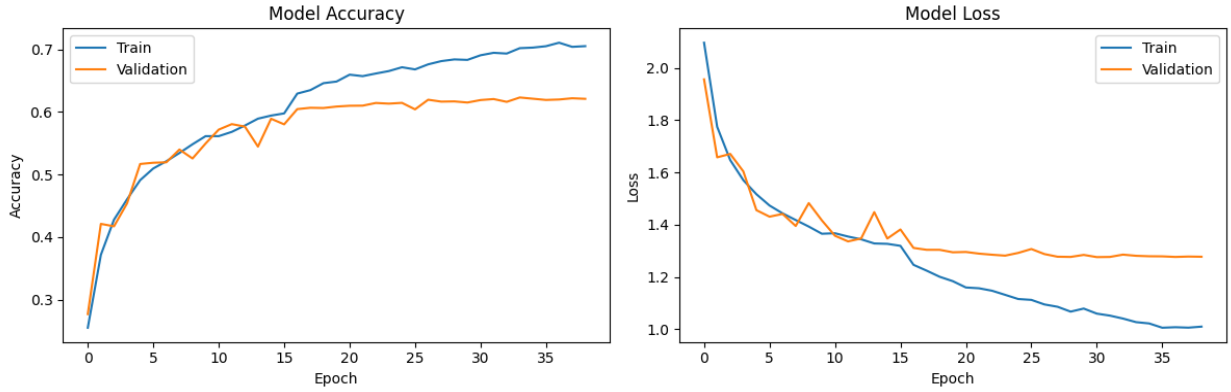
4.2 Przebieg Eksperymentów

Warto zwrócić uwagę, iż początkowo została przyjęta identyfikacja **siedmiu** emocji. Były to radność, złość, smutek, neutralny wyraz twarzy, strach, zaskoczenie, zniesmaczenie (ang. happy, angry, sad, neutral, fear, surprise, disgust). Nie został określony limit danych, co okazało się kluczącym czynnikiem.

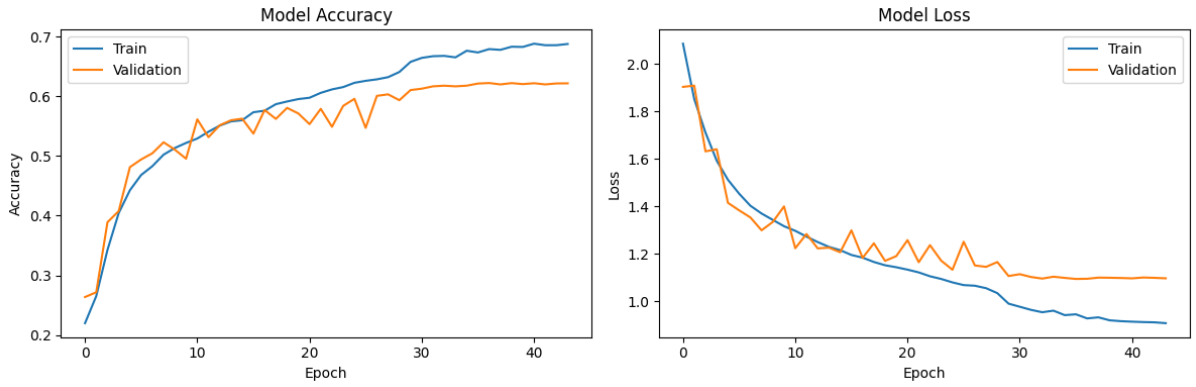
Parametry	***Model pierwszy***	***Model drugi***
Warstwy CNN	3 warstwy (32, 64, 128)	3 warstwy (32, 64, 128,256)
Warstwy Dense *(liczba neuronów)*	512, 256, 128	256, 128, 128
Optymalizator	Adam	Adam
Learn Rate*	0,0001	0,000001
Accuracy*	61%	63%
przerwanie* /liczba epok	39/50	44/50

Tabela 2 Tabela reprezentująca przykładowe parametry podczas trenowania modelu

**przerwanie- model zatrzymywał się, gdy pojawiało się przeuczanie*



Rysunek 5 Wynik dla modelu 1, accuracy 61%



Rysunek 6 Wynik dla modelu 2, accuracy 63%

Kolejnym krokiem po nieudanych eksperymentach było organicznie się do analizy trzech emocji ‘sad’ i ‘happy’ i ‘angry’ Dodatkowo równocześnie z ta zmianą zmieniano optymalizator z **Adam** na **AdamW**. W rezultacie otrzymano znacząco lepsze wyniki.

Zmiana optymalizatora wynikała z problemu braku kontrolownia wag w pierwotnym optymalizatorze. AdamW skuteczniej zapobiega problemom z przeuczeniem oraz niestabilnością uczenia. W Adamie regularyzacja L2 (kara za duże wagi) jest dodawana do gradientów w trakcie ich obliczania. To oznacza, że gradienty są "modyfikowane" regularyzacją, zanim zostaną użyte do aktualizacji wag. Natomiast w AdamieW regularyzacja L2 jest **oddzielona** od gradientów i stosowana **bezpośrednio do wag**.

Poniżej znajdują się wzory odpowiadające za sposób aktualizacji parametrów modelu θ(t) w algorytmie Adam oraz AdamW.

Wzór reprezentujący optymalizator Adam:

$$\theta = \theta^{(t-1)} - \alpha^t * \frac{\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}+\varepsilon}}$$

- $\boldsymbol{\theta^{(t)}}$ – parametry modelu w czasie t, który jest aktualizowany,
- $\boldsymbol{\theta^{(t-1)}}$ – Parametry modelu w czasie t−1, czyli poprzednia wersja parametrów,
- $\boldsymbol{\alpha^t}$ – Współczynnik uczenia (learning rate) w czasie t. Jest to skalar, który kontroluje wielkość kroków aktualizacji.
- $\boldsymbol{\hat{m}^{(t)}}$ – Wygładzony moment pierwszego rzędu w czasie t. Jest to średnia z gradientów, obliczona w sposób wygładzający, co pomaga w stabilizacji procesu optymalizacji
- $\boldsymbol{\hat{v}^{(t)}}$ – Wygładzony moment drugiego rzędu w czasie t. To średnia kwadratów gradientów, która informuje o tym, jak duże są zmiany gradientów.
- **ϵ** – Mała stała dodana do mianownika w celu uniknięcia dzielenia przez zero. Zwykle ma wartość rzędu 10^{-8}

Wzór reprezentujący optymalizator AdamW(Adam with Weight Decay):

$$\theta = \theta^{(t-1)} - \alpha^t * \frac{\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)} + \varepsilon}} - \alpha^t \lambda \theta^{(t-1)}$$

- λ to współczynnik regularyzacji L2
- pozostałe symbole mają takie samo znaczenie, jak w algorytmie Adam

Można zauważyć, że w worze reprezentującym drugi optymalizator pojawia się dodatkowy człon. Odpowiada on za ‘przycinanie’ wag, aby móc zapobiec ich nadmiernemu wzrostowi. Jest on stosowany bezpośrednio do wag, regularyzacja jest oddzielna i dodana na końcu jako osobny krok.

Parametry	*Model pierwszy*	*Model drugi*	*Model trzeci*	*Model czwarty*
Warstwy CNN	32, 64, 128, 128	32, 64, 128, 128	32, 64, 128, 128	32, 64, 128, 128
Warstwy Dense *(liczba neuronów)*	64, 128, 128	64, 128, 128	64, 128, 128	64, 128, 128
Optymalizator (learning_rate)	AdamW(0.001)	Adam	AdamW (0.0005)	AdamW (0.0005)
Learn Rate*	0.001	0.001	0.0000001	0.0000003
Accuracy*	75%	71%	73%	73%

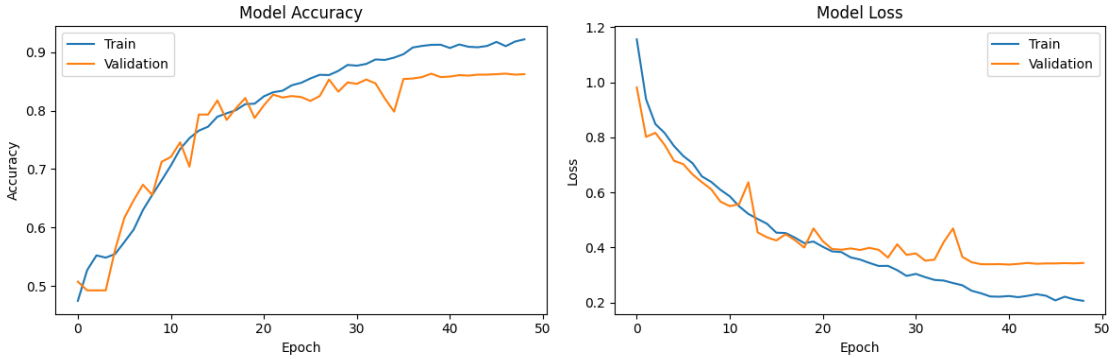
Kolejnym korkiem, aby uzyskać lepszy wyniki było organicznie się do 2 emocji ‘happy’ i ‘sad’, ponieważ zdjęcia nie są standaryzowane jak fotografie paszportowe, występuje duża zmienność w sposobie wyrażania emocji. Okazało się to być kluczowym działaniem ponieważ, wynik dokładności znacząco się poprawił z 70% wzrósł na 87%, przy następujących parametrach:

Parametry	*Model pierwszy*	*Model drugi*	*Model trzeci*
Warstwy CNN	4 warstwy (32, 64, 128, 256)	4 warstwy (32, 64, 128, 256)	4 warstwy (32, 64, 128, 256)
Warstwy Dense *(liczba neuronów)*	150, 120, 70	160,125,80	170,135,85
Optymalizator (learning_rate)	AdamW (0.0005)	AdamW (0.0005)	AdamW (0.0005)
Learn Rate*	0,000002	0,000002	0,000002
Accuracy*	86%	87%	84%

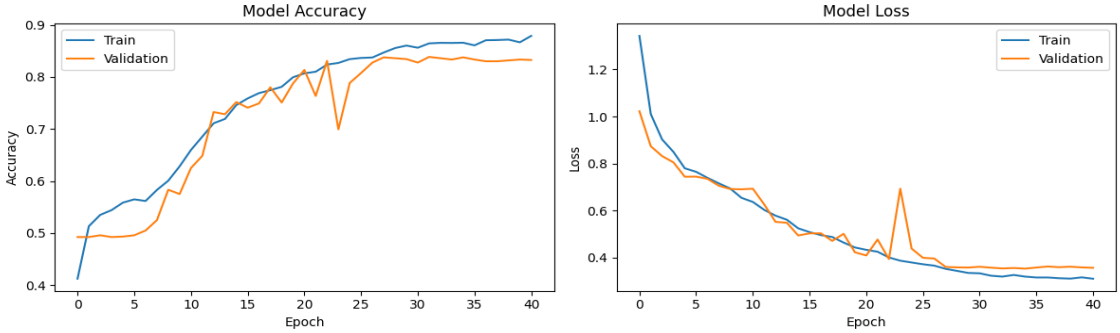
4.3 Wyniki i Analiza

Proces uczenia modelu można przeanalizować na podstawie wykresów przedstawiających dokładność (accuracy) oraz funkcję straty (loss) w czasie treningu. Model drugi osiągnął najlepszą skuteczność na poziomie 87%, co ilustruje Rysunek 7. Krzywa uczenia pokazuje stabilny wzrost dokładności, osiągając około 93% na zbiorze treningowym i 87% na walidacyjnym. Funkcja straty systematycznie maleje, stabilizując się na poziomie 0.2-0.4, co świadczy o dobrym dopasowaniu modelu.

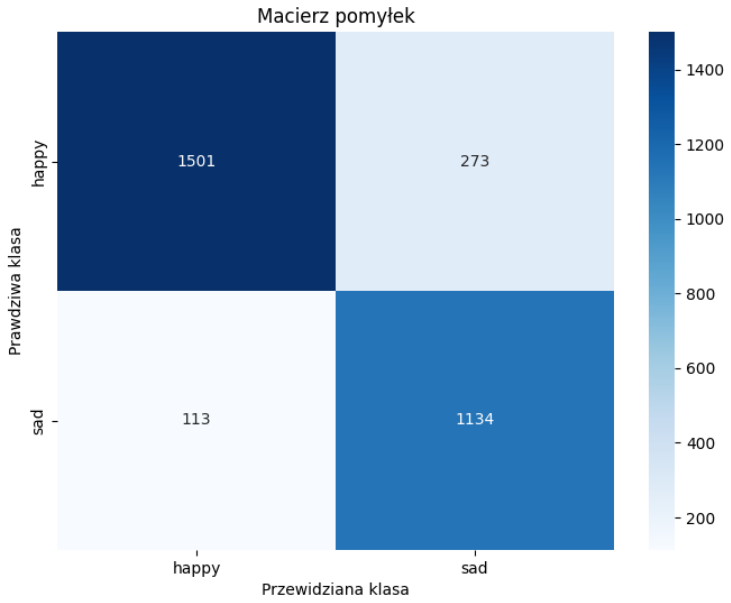
Dla porównania, model trzeci (Rysunek 8) osiągnął nieco niższą skuteczność 84%. Charakteryzuje się on większymi wahaniami w procesie uczenia, co widać szczególnie w okolicy 20-25 epoki. Niemniej jednak, mniejsza różnica między dokładnością treningową a walidacyjną sugeruje dobre właściwości generalizacyjne.



Rysunek 7 Wynik dla modelu 2, accuracy 87%



Rysunek 8 Wynik dla modelu trzeciego accuracy 84%



Rysunek 9 Macierz pomyłek dla modelu drugiego, accuracy 87%

Skuteczność klasyfikacji najlepiej obrazuje macierz pomyłek (Rysunek 9). Model poprawnie sklasyfikował 1501 przypadków szczęśliwych i 1134 smutnych, przy stosunkowo niewielkiej liczbie błędnych klasyfikacji (273 dla happy i 113 dla sad). Zbalansowane wyniki dla obu klas emocji

potwierdzają stabilność i niezawodność modelu w rozpoznawaniu zarówno pozytywnych, jak i negatywnych ekspresji twarzy.

5. Kod Projektu.

Kod rozpoczyna się od parametryzacji wartości:

```
IMG_SIZE = 48
NUM_CHANNELS = 1
NUM_CLASSES = 3
BATCH_SIZE = 64
EPOCHS = 1000
LEARN_RATE = 0.000002
```

Zdjęcie 1 Parametry kodu

Następnie wywoływana jest funkcja główna __main__:

```
if __name__ == "__main__":
    db_path = './photos'
    model, history = train_model(db_path)

    plot_training_history(history)
    model.save('emotion_recognition_model.keras')

    # Uzyskanie accuracy
    test_images, test_labels = load_dataset(os.path.join(db_path, 'test'))
    test_images, test_labels = preprocess_data(test_images, test_labels)
    _, accuracy = model.evaluate(test_images, test_labels)

    # Zapis modelu
    model_name = f'model_{accuracy:.2f}.keras'
    model.save(model_name)
    print(f"Model saved as: {model_name}")



    # Test pre-trained model
    model_path = 'model_0.87.keras'
    test_sample_images(model_path, os.path.join(db_path, 'test'))
```

Zdjęcie 2 Funkcja główna programu

Tak jak opisano kod na zdjęciu, na początku wywoływania jest funkcja train_model(). Następnie wykonywane są testy, a model jest zapisywany, po czym zapisany model jest kolejny raz testowany.

Funkcja train_model() wygląda następująco:

```python
def train_model(db_path):
    print("Loading training data...")
    train_images, train_labels = load_dataset(os.path.join(db_path, 'train'))
    print(f"Loaded {len(train_images)} training images")

    print("\nLoading test data...")
    test_images, test_labels = load_dataset(os.path.join(db_path, 'test'))
    print(f"Loaded {len(test_images)} test images")

    # Preprocess data
    train_images, train_labels = preprocess_data(train_images, train_labels)
    test_images, test_labels = preprocess_data(test_images, test_labels)

    train_images, val_images, train_labels, val_labels = train_test_split(
        train_images, train_labels, test_size=0.2, random_state=42
    )

    print("\nCreating model...")
    model = create_model()
    model.compile(
        optimizer=tf.keras.optimizers.AdamW(learning_rate=0.0005, weight_decay=0.004),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )


    model.summary()
```

Zdjęcie 3 Funkcja odpowiadająca za trenowanie modelu.

[…] – Funkcja zbyt długa, by zmiescic ją na screenie.

Przygotowywanie danych odbywa się przez funkcję preprocess_data():

```python
def preprocess_data(images, labels):
    """
    Normalize images and convert labels to one-hot encoding
    """
    images = images.astype('float32') / 255.0

    labels = tf.keras.utils.to_categorical(labels, NUM_CLASSES)

    return images, labels
```

Zdjęcie 4 Funkcja odpowiadająca za przygotowanie danych.

Architektura modelu tworzona jest funkcją create_model():

```python
def create_model():
    inputs = Input(shape=(IMG_SIZE, IMG_SIZE, NUM_CHANNELS))


    x = layers.Conv2D(32, (3, 3), activation='relu',
                      kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE))(inputs
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Dropout(0.3)(x)

    x = layers.Conv2D(64, (3, 3), activation='relu',
                      kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE))(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Dropout(0.3)(x)

    x = layers.Conv2D(128, (3, 3), activation='relu',
                      kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE))(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Dropout(0.3)(x)

    x = layers.Conv2D(256, (3, 3), activation='relu',
                      kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE))(x)
```

Zdjęcie 5 Funkcja tworzenia modelu

```python
x = layers.Conv2D(256, (3, 3), activation='relu',
                  kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)

x = layers.Flatten()(x)
x = layers.Dense(160, use_bias=False, kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE), name='dense_
x = layers.BatchNormalization(name='bn_1')(x)
x = layers.ReLU()(x)
x = layers.Dropout(0.4)(x)

x = layers.Dense(125, use_bias=False, kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE), name='dense_
x = layers.BatchNormalization(name='bn_2')(x)
x = layers.ReLU()(x)
x = layers.Dropout(0.4)(x)

x = layers.Dense(80, use_bias=False, kernel_regularizer=tf.keras.regularizers.l2(LEARN_RATE), name='dense_3
x = layers.BatchNormalization(name='bn_3')(x)
x = layers.ReLU()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(NUM_CLASSES, activation='softmax')(x)

return models.Model(inputs=inputs, outputs=outputs)
```

Zdjęcie 6 Funkcja tworzenia modelu część 2

W kodzie wyraźnie widoczne są 4 warstwy konwolucyjne oraz 3 warstwy gęste.

Za wczytywanie danych odpowiada funkcja load_dataset():

```
def load_dataset(base_path):
    images = []
    labels = []
    emotion_map = {
         'sad': 1, 'happy': 0,
    }

    # Inicjalizacja globalnych wartości
    global_min = float('inf')
    global_max = float('-inf')

    for emotion in emotion_map.keys():
        path = os.path.join(base_path, emotion)
        if not os.path.exists(path):
            print(f"Warning: Path {path} does not exist")
            continue

        print(f"Loading {emotion} images from {path}")
        counter = 0
        limit = 3000
        if base_path == './photos\test':
            limit = 950

        for img_path in os.listdir(path):
            counter += 1
            if counter >= limit:
                counter = 0
                break
```

Zdjęcie 7 Funkcja odpowiadająca za wczytywanie danych.

6. Analiza Wyników i Wnioski

5.1 Ocena Wyników

Na tym etapie do oceny jakości modelu wykorzystano folder testowy zawierający zbiór zdjęć dla różnych emocji. Każda kategoria emocjonalna została poddana analizie przez wytrenowany wcześniej model. W przedstawionym przypadku widoczne są wyniki dla emocji "happy", gdzie model osiągnął 80% skuteczności klasyfikacji (8/10 poprawnych predykcji). Warto zauważyć, że model generuje również poziom pewności (confidence score) dla każdej predykcji, co pozwala ocenić wiarygodność klasyfikacji. Dla większości poprawnych klasyfikacji model wykazał wysoką pewność (>0.95), podczas gdy w przypadku błędnych klasyfikacji poziomy pewności były zróżnicowane (0.52 i 0.93).

Rysunek 10 Predykcja modelu na danych testowych

Rysunek 11 Predykcja modelu na danych testowych

5.2 Wnioski

- Aby model działał poprawnie, konieczne było odpowiednie przygotowanie zdjęć. Polegało to na: zamianie zdjęć na czarno-białe, ujednoliceniu ich rozmiaru do 48x48 pikseli, przekształceniu wartości pikseli na zakres od 0 do 1, odpowiednim oznaczeniu klas emocji (szczęście i smutek).
- Zastosowanie optymalizatora AdamW zamiast Adam poprawiło stabilność i skuteczność modelu. AdamW pozwolił na lepsze kontrolowanie wag w trakcie uczenia, co zapobiegło błędom wynikającym z przeuczenia.
- Dane użyte w projekcie były różnorodne – zdjęcia przedstawiały twarze z różnych kątów, czasami były zasłonięte, a niektóre miały błędne etykiety. Skupienie się na dwóch emocjach (szczęściu i smutku) pomogło uzyskać lepsze wyniki, poprawiając dokładność modelu.
- Najlepsze rezultaty uzyskano dzięki użyciu 4 warstw konwolucyjnych i 3 warstw gęstych. Model został dobrze zaprojektowany, z funkcjami zapobiegającymi przeuczeniu, co pozwoliło osiągnąć dokładność 87%.
- Ważne było wprowadzenie mechanizmów, takich jak wczesne zatrzymanie (Early Stopping), aby przerwać uczenie, gdy wyniki przestawały się poprawiać oraz stopniowe zmniejszanie tempa uczenia (ReduceLROnPlateau), aby lepiej dostosować proces trenowania.
- Model dobrze rozpoznaje emocje szczęścia i smutku, osiągając dobrą dokładność i równowagę w klasyfikacji obu emocji. Stabilne wyniki na danych testowych pokazują, że model ma potencjał do zastosowań w praktyce.