

**Wydział Zarządzania**

**Sieci neuronowe i uczenie głębokie**

***„Analiza i klasyfikacja stresu na podstawie ekspresji twarzy z wykorzystaniem CNN”***

Jakub Le Van

Spis treści

[**1.** **Wstęp** 3](#_Toc188721546)

[**2.** **Metodologia** 3](#_Toc188721547)

[**2.1.** **Cel projektu** 3](#_Toc188721548)

[**2.2.** **Użyte biblioteki i narzędzia** 3](#_Toc188721549)

[**2.3.** **Opis danych** 4](#_Toc188721550)

[**2.4.** **Podgląd danych** 4](#_Toc188721551)

[**3.** **Implementacja** 5](#_Toc188721552)

[**3.1.** **Definicja modelu** 5](#_Toc188721553)

[**3.2.** **Proces treningu** 6](#_Toc188721554)

[**3.2.1.** **Trening modelu** 6](#_Toc188721555)

[**3.2.2.** **Wizualizacja wyników** 6](#_Toc188721556)

[**3.3.** **Proces walidacji** 8](#_Toc188721557)

[**3.3.1.** **Wyniki testów** 8](#_Toc188721558)

[**3.4.** **Proces testowania** 9](#_Toc188721559)

[**4.** **Podsumowanie** 10](#_Toc188721560)

# **Wstęp**

Rozpoznawanie stresu na podstawie obrazu twarzy to problem, który zyskuje na znaczeniu w obszarze sztucznej inteligencji. Subtelne zmiany w mimice, napięcie mięśni czy wyraz oczu mogą świadczyć o podwyższonym poziomie napięcia emocjonalnego. Wykorzystanie takich informacji ma zastosowanie m.in. w opiece zdrowotnej, psychologii czy systemach bezpieczeństwa.

Celem projektu było opracowanie modelu do detekcji stresu na twarzach z wykorzystaniem konwolucyjnych sieci neuronowych (CNN). Przy pomocy frameworka TensorFlow zaprojektowano, wytrenowano i przetestowano model na danych obrazowych, analizując wpływ architektury sieci i parametrów treningu na skuteczność klasyfikacji.

Konwolucyjne sieci neuronowe są szeroko stosowane w analizie obrazów dzięki ich zdolności do wykrywania wzorców na różnych poziomach szczegółowości. W projekcie uwzględniono wyzwania związane z różnorodnością danych, takie jak zróżnicowanie wyrazów twarzy czy warunki oświetleniowe, które mogą wpływać na jakość klasyfikacji.

# **Metodologia**

## **Cel projektu**

Celem projektu było opracowanie modelu wykorzystującego konwolucyjne sieci neuronowe (CNN) do klasyfikacji danych obrazowych. Model miał na celu rozpoznawanie obiektów na obrazach, a jego skuteczność została oceniona za pomocą standardowych metryk, takich jak dokładność, precyzja i czułość.

## **Użyte biblioteki i narzędzia**

W projekcie wykorzystałem kilka popularnych bibliotek, które znacząco ułatwiły pracę zarówno przy przetwarzaniu danych, jak i budowie modelu sieci neuronowej. Oto najważniejsze z nich:

* **TensorFlow** – Główne narzędzie używane do stworzenia i trenowania modelu konwolucyjnej sieci neuronowej. Dzięki TensorFlow mogłem w łatwy sposób zdefiniować architekturę sieci, skonfigurować funkcję kosztu oraz wybrać odpowiedni optymalizator i metryki.
* **NumPy** – Ta biblioteka była nieoceniona przy przetwarzaniu danych, szczególnie przy manipulacji tablicami numerycznymi. Użyłem jej m.in. do normalizacji wartości pikseli obrazów.
* **Matplotlib** – Służyła mi do wizualizacji wyników. Dzięki niej mogłem np. zobaczyć kilka przykładowych obrazów z zestawu danych, a także stworzyć wykresy przedstawiające dokładność i stratę modelu w czasie.
* **OpenCV** – Wykorzystałem tę bibliotekę do wstępnego przetwarzania obrazów. Użyłem jej, żeby zmienić rozdzielczość zdjęć na 256x256 pikseli i przygotować je do analizy przez model.

## **Opis danych**

Dane wykorzystane w projekcie zostały pobrane z platformy Kaggle, z następującego zestawu danych: Stress and Non-Stress Images (https://www.kaggle.com/datasets/preritbhagat/stress-non-stress-images/data).

Zbiór ten zawiera obrazy przeznaczone do klasyfikacji binarnej: obrazy osób w stanie stresu i bez stresu.

* **Typ danych**: Obrazy.
* **Format**: RGB.
* **Rozdzielczość**: Obrazy zostały przeskalowane do wymiarów 256x256 pikseli.
* **Liczność danych**: 4409 zdjęć należących do 2 klas.

Zestaw danych został podzielony na trzy części:

* **Zbiór treningowy**: 70% danych, wykorzystywany do uczenia modelu.
* **Zbiór walidacyjny**: 20% danych, wykorzystywany do oceny modelu w trakcie treningu.
* **Zbiór testowy**: 10% danych, przeznaczony do końcowej oceny wydajności modelu.

1. train\_size = int(len(data) \* 0.7)

2. val\_size = int(len(data) \* 0.2)

3. test\_size = int(len(data) \* 0.1)

4.

5. train = data.take(train\_size)

6. val = data.skip(train\_size).take(val\_size)

7. test = data.skip(train\_size + val\_size).take(test\_size)

8.

## **Podgląd danych**

Aby lepiej zrozumieć strukturę i charakterystykę danych, wyświetliłem kilka próbek obrazów z zestawu danych. Poniższa wizualizacja przedstawia cztery przykładowe obrazy z różnych klas. Klasa każdego obrazu jest oznaczona nad nim jako 1 (stres) lub 0 (brak stresu).

Obrazy przedstawiają różne osoby w różnych pozach i z różnymi wyrazami twarzy.

Obraz zawierający Ludzka twarz, zrzut ekranu, skóra, osoba

Opis wygenerowany automatycznie

1. batch = data\_iterator.next()

2.

3. fig, ax = plt.subplots(ncols=4, figsize=(20, 20))

4. for idx, img in enumerate(batch[0][:4]):

5. ax[idx].imshow(img.astype(int))

6. ax[idx].title.set\_text(batch[1][idx])

7.

# **Implementacja**

## **Definicja modelu**

Model oparty na konwolucyjnych sieciach neuronowych (CNN) został zbudowany w TensorFlow. Składa się z trzech warstw konwolucyjnych z aktywacją ReLU i warstw MaxPooling, które zmniejszają wymiar danych. Na końcu zastosowano warstwy Flatten i Dense, a wyjście modelu to neuron z aktywacją sigmoid, odpowiedzialny za klasyfikację binarną.

1. model.add(Conv2D(16, (3,3), 1, activation='relu', input\_shape=(256,256,3)))

2. model.add(MaxPooling2D())

3.

4. model.add(Conv2D(32, (3,3), 1, activation='relu'))

5. model.add(MaxPooling2D())

6.

7. model.add(Conv2D(16, (3,3), 1, activation='relu'))

8. model.add(MaxPooling2D())

9.

10. model.add(Flatten())

11.

12. model.add(Dense(256, activation='relu'))

13. model.add(Dense(1, activation='sigmoid'))

14.

* **Warstwy konwolucyjne** - Warstwy te odpowiadają za wyodrębnianie cech obrazu, takich jak krawędzie, tekstury czy inne wzorce.
* **Warstwy poolingowe** - Po każdej warstwie konwolucyjnej zastosowano warstwę poolingową, która redukuje rozmiar map cech.
* **Warstwa Flatten** - Po przetworzeniu obrazu przez warstwy konwolucyjne i poolingowe dane mają postać wielowymiarowych map cech. Warstwa Flatten zamienia te dane na jednowymiarowy wektor, który można przetwarzać w warstwach gęstych.
* **Warstwy Dense** - Te warstwy to w pełni połączone neurony, które na podstawie wyodrębnionych cech podejmują decyzję o klasyfikacji.

Do optymalizacji użyłem algorytmu Adam, a jako funkcję kosztu wybrałem Binary Crossentropy. Model monitorował dokładność w zbiorach treningowym i walidacyjnym.

1. model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])

## **Proces treningu**

### **Trening modelu**

Model trenowałem przez 20 epok, monitorując jego działanie na zbiorze treningowym i walidacyjnym. Skupiałem się głównie na dwóch wskaźnikach:

* **Strata (loss)** – pokazuje, jak dobrze model radzi sobie z dopasowaniem do danych.
* **Dokładność (accuracy)** – mierzy procent poprawnych klasyfikacji.

Podczas treningu model szybko osiągnął wysoką dokładność. Na koniec 20. epoki dokładność na zbiorze treningowym wyniosła **99,1%**, a na walidacyjnym **92,3%**. Strata treningowa spadła do **0,0249**, co wskazuje na dobre dopasowanie modelu do danych treningowych.

1. hist = model.fit(train,

2. epochs=20,

3. validation\_data=val,

4. callbacks=[tensorboard\_callback])

### **Wizualizacja wyników**

Aby lepiej zobrazować postęp treningu, stworzyłem dwa wykresy – strat i dokładności – dla zbiorów treningowego i walidacyjnego.

* **Wykres strat (loss)** - Na wykresie widać, że strata na zbiorze treningowym systematycznie maleje, co jest oznaką skutecznego treningu. Strata walidacyjna również spada na początku, ale w końcowych epokach zaczyna lekko rosnąć, co może wskazywać na początek przeuczenia.

1. fig = plt.figure()

2. plt.plot(hist.history['loss'], color='teal', label='loss')

3. plt.plot(hist.history['val\_loss'], color='orange', label='val\_loss')

4. fig.suptitle('Loss', fontsize=20)

5. plt.legend(loc="upper left")

6. plt.show()

Obraz zawierający tekst, diagram, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

* **Wykres dokładności (accuracy)** - Widać, że dokładność modelu na zbiorze treningowym bardzo szybko rośnie i stabilizuje się powyżej 99%. Dokładność na walidacji jest nieco niższa, ale osiąga solidny wynik powyżej 92%.

1. fig = plt.figure()

2. plt.plot(hist.history['accuracy'], color='teal', label='accuracy')

3. plt.plot(hist.history['val\_accuracy'], color='orange', label='val\_accuracy')

4. fig.suptitle('Accuracy', fontsize=20)

5. plt.legend(loc="upper left")

6. plt.show()

Obraz zawierający tekst, diagram, zrzut ekranu, Wykres

Opis wygenerowany automatycznie

## **Proces walidacji**

### **Wyniki testów**

Na zbiorze testowym sprawdziłem działanie modelu za pomocą trzech metryk:

* **Precyzja (Precision)** – pokazuje, ile z próbek zaklasyfikowanych jako "stres" faktycznie należy do tej klasy.
* **Czułość (Recall)** – wskazuje, jaki procent próbek z klasy "stres" został poprawnie wykryty.
* **Dokładność (Binary Accuracy)** – ogólny odsetek poprawnych klasyfikacji.

1. from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy

2.

3. pre = Precision()

4. re = Recall()

5. acc = BinaryAccuracy()

6.

7. for batch in test.as\_numpy\_iterator():

8. X, y = batch

9. yhat = model.predict(X)

10. pre.update\_state(y, yhat)

11. re.update\_state(y, yhat)

12. acc.update\_state(y, yhat)

13.

14. print(pre.result().numpy(), re.result().numpy(), acc.result().numpy())

**Wyniki testów:**

* **Precyzja:** 92,63%
* **Czułość:** 89,33%
* **Dokładność:** 90,38%

Model osiągnął bardzo dobre wyniki, szczególnie jeśli chodzi o precyzję, co oznacza, że większość obrazów oznaczonych jako "stres" była poprawna. Czułość również jest wysoka, co pokazuje, że model potrafi wykryć większość próbek "stresu", chociaż czasami może je przeoczyć.

## **Proces testowania**

Obraz zawierający Ludzka twarz, uśmiech, zrzut ekranu, okulary

Opis wygenerowany automatycznieModel został przetestowany na obrazie, który nie był częścią zbioru treningowego ani testowego. Proces przebiegał następująco:

* **Wczytanie i przeskalowanie obrazu** - Obraz załadowano za pomocą OpenCV, a następnie zmieniono jego rozmiar na 256x256 pikseli i przeskalowano wartości pikseli do przedziału [0, 1], aby spełniał wymagania modelu.
* **Przewidywanie klasy** - Model przewidział prawdopodobieństwo przynależności obrazu do klasy "stres", zwracając wartość 0.99999744, co wyraźnie wskazuje na klasę "stres".
* **Wynik** - Na podstawie progu 0.5 model zaklasyfikował obraz jako "stres".

Model poprawnie sklasyfikował nowy obraz, co potwierdza jego zdolność do generalizacji. Zdjęcie zostało wykonane przeze mnie w sytuacji, gdy, mimo uśmiechu, mogłem wykazywać oznaki stresu – zdjęcie zrobiono w czasie poprzedzającym sesję egzaminacyjną. Wynik pokazuje, że model skutecznie analizuje nowe dane, nawet w złożonych kontekstach emocjonalnych.

1. import cv2

2.

3. img = cv2.imread('test/test3.jpg')

4. plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

5. plt.show()

6.

7. resize = tf.image.resize(img, (256,256))

8. plt.imshow(resize.numpy().astype(int))

9. plt.show()

10.

11. yhat = model.predict(np.expand\_dims(resize/255, 0))

12.

13. yhat

14.

15. if yhat > 0.5:

16. print(f'Przewidywana klasa to stres')

17. else:

18. print(f'Przewidywana klasa to brak stresu')

# **Podsumowanie**

Model oparty na konwolucyjnych sieciach neuronowych (CNN) osiągnął bardzo dobre wyniki w klasyfikacji obrazów. Na zbiorze testowym dokładność wyniosła 90,38%, precyzja 92,63%, a czułość 89,33%. To pokazuje, że model skutecznie rozróżnia obrazy należące do klas "stres" i "brak stresu".

Testowanie na nowym obrazie również potwierdziło, że model dobrze generalizuje i potrafi analizować dane, które nie były częścią zbioru treningowego. Dzięki temu można rozważać jego praktyczne zastosowanie, np. w systemach do analizy emocji lub wykrywania stresu.

**Najważniejsze wnioski:**

* Model jest skuteczny w rozpoznawaniu obrazów z klas "stres" i "brak stresu".
* Wyniki testów pokazują, że model może być przydatny w praktyce, szczególnie w zadaniach takich jak analiza emocji czy monitorowanie stresu.
* Precyzja i czułość są na zbliżonym poziomie, co wskazuje na zrównoważoną wydajność modelu.