**Programowanie logiczne w sztucznej inteligencji**

Michał Spyrzewski nr. 7102

VI sem. Informatyka

**Wykrywanie emocji w czasie rzeczywistym**

# **1.Wprowadzenie do projektu**

### **Geneza Pracy**

Wybór tematu dotyczącego wykrywania emocji za pomocą modelu głębokiego uczenia i aplikacji Tkinter wynika z rosnącego zainteresowania i znaczenia analizy emocji w różnych dziedzinach. W dzisiejszym społeczeństwie, technologia odgrywa kluczową rolę w komunikacji i interakcji międzyludzkiej.

**Zastosowanie**

Rozpoznawanie emocji ma szerokie zastosowanie w wielu obszarach, takich jak:

1. Interakcje Człowiek-Komputer: Tworzenie bardziej naturalnych i intuicyjnych interfejsów użytkownika.
2. Edukacja: Monitorowanie stanu emocjonalnego uczniów, aby lepiej dostosować metody nauczania.
3. Medycyna: Wsparcie dla osób z zaburzeniami emocjonalnymi i monitorowanie stanu psychicznego pacjentów.
4. Bezpieczeństwo Publiczne: Identyfikacja potencjalnie niebezpiecznych zachowań na podstawie analizy emocji.
5. Marketing i Reklama: Analiza reakcji emocjonalnych konsumentów na produkty i reklamy.

Temat ten został wybrany również ze względu na rozwój technologii głębokiego uczenia, która oferuje nowe możliwości w analizie obrazów i rozpoznawaniu wzorców. Dzięki temu można skutecznie identyfikować różne emocje na podstawie obrazów twarzy, co było trudne do osiągnięcia przy użyciu tradycyjnych metod.

### **Cel i Zakres Pracy**

Celem pracy jest zaprojektowanie i implementacja systemu do rozpoznawania emocji na podstawie obrazów twarzy w czasie rzeczywistym przy użyciu modelu głębokiego uczenia oraz aplikacji stworzonej w Tkinter. System ten ma na celu:

1. Zidentyfikowanie różnych emocji: Rozpoznawanie emocji takich jak gniew, obrzydzenie, strach, radość, neutralność, smutek i zaskoczenie.
2. Realizacja w czasie rzeczywistym: Umożliwienie przetwarzania strumienia wideo na żywo z kamerki internetowej lub innego źródła wideo.
3. Integracja z interfejsem użytkownika: Stworzenie intuicyjnego interfejsu użytkownika za pomocą biblioteki Tkinter, który umożliwia łatwą obsługę systemu.

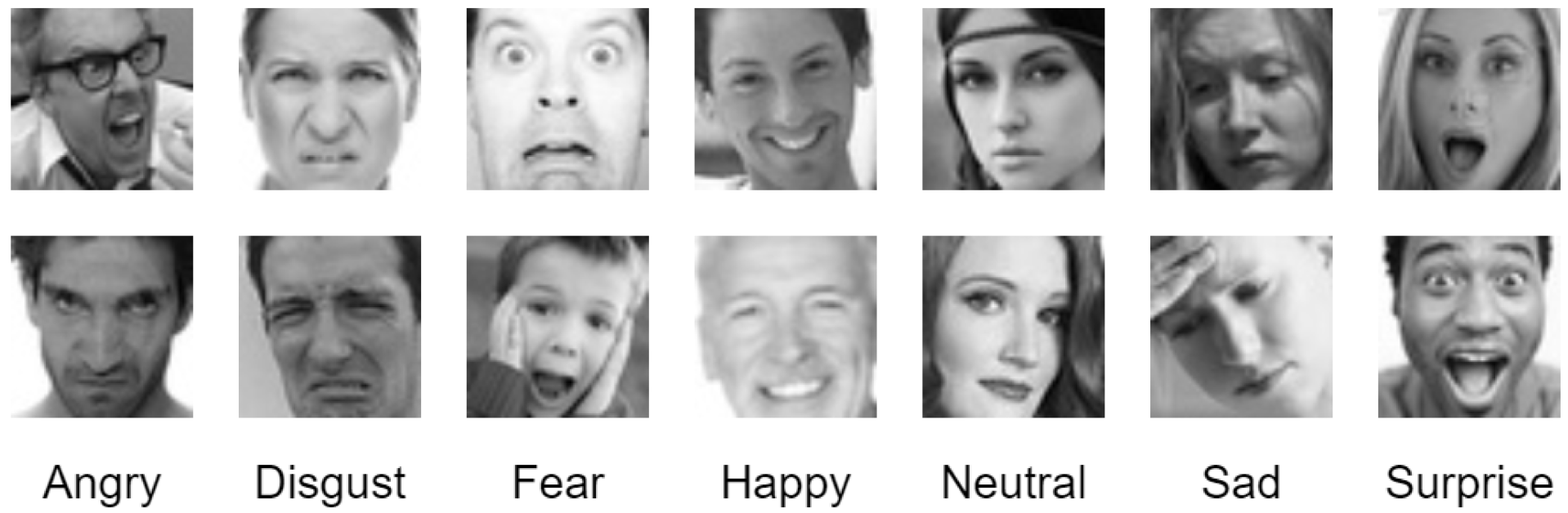
#### **Zakres Pracy**

Zakres pracy obejmuje następujące etapy:

1. Przygotowanie zbioru danych: Wybór odpowiedniego zbioru danych do trenowania modelu oraz jego przetworzenie i przygotowanie.
2. Projektowanie modelu: Zdefiniowanie architektury modelu głębokiego uczenia, który będzie używany do klasyfikacji emocji.
3. Trenowanie modelu: Trenowanie modelu na przygotowanych danych oraz ocena jego wydajności.
4. Implementacja aplikacji: Stworzenie aplikacji w Tkinter, która będzie używać wytrenowanego modelu do rozpoznawania emocji w czasie rzeczywistym.
5. Testowanie i walidacja: Przeprowadzenie testów, aby sprawdzić dokładność i wydajność systemu w różnych warunkach.
6. Dokumentacja i prezentacja wyników: Przygotowanie dokumentacji projektu oraz prezentacja wyników i wniosków.

Realizacja tych celów i zakresu pracy pozwoli na stworzenie kompletnego systemu do rozpoznawania emocji, który może być wykorzystany w praktycznych zastosowaniach, a także stanowić podstawę dla przyszłych badań i rozwoju w tej dziedzinie.

**Dane wykorzystywane do Treningu modelu**

****

#### **Opis Zbioru Danych**

Do trenowania modelu głębokiego uczenia, który ma za zadanie rozpoznawać emocje na podstawie obrazów twarzy, wykorzystano zestaw danych zawierający obrazy twarzy oznaczone odpowiednimi etykietami emocji. Wybrany zbiór danych powinien spełniać następujące kryteria:

1. Różnorodność Emocji: Dane powinny obejmować obrazy twarzy przedstawiające różne emocje, takie jak gniew, obrzydzenie, strach, wesołość, neutralność, smutek i zaskoczenie.
2. Różnorodność Demograficzna: Obrazy powinny obejmować różne grupy demograficzne (np. różne wieki, płcie, rasy), aby model mógł dobrze generalizować na nowe, nieznane dane.
3. Jakość Obrazów: Obrazy powinny mieć wystarczająco dobrą jakość, aby możliwe było wyodrębnienie istotnych cech twarzy.

#### **Wybrany Zbiór Danych**

W projekcie wykorzystano jeden z popularnych zbiorów danych do rozpoznawania emocji FER-2013. Zbiór FER-2013 spełnia wszystkie powyższe kryteria i jest powszechnie używany w badaniach nad rozpoznawaniem emocji.

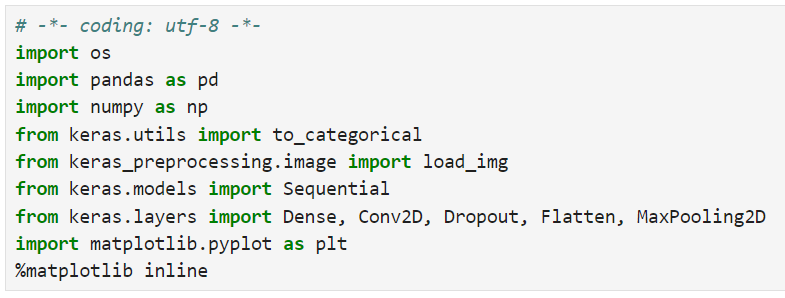
#### **Struktura Zbioru Danych**

Zbiór danych FER-2013 składa się z obrazów w odcieniach szarości o wymiarach 48x48 pikseli. Każdy obraz jest przypisany do jednej z siedmiu kategorii emocji. Zbiór danych jest podzielony na dwie główne części:

1. Zbiór Treningowy (Training Set):
   * Zawiera obrazy wykorzystywane do trenowania modelu.
   * Każdy obraz ma przypisaną etykietę emocji.
   * Rozmiar: Około 28,000 obrazów.
2. Zbiór Walidacyjny (Validation Set):
   * Zawiera obrazy wykorzystywane do oceny wydajności modelu podczas treningu.
   * Każdy obraz ma przypisaną etykietę emocji.
   * Rozmiar: Około 7,000 obrazów.

# **2. Przygotowanie środowiska jupyter-notebook**

**Najpierw importujemy niezbędne biblioteki. Każda z nich pełni określoną rolę w przetwarzaniu danych, budowie i trenowaniu modelu oraz wizualizacji wyników.**

****

**os:** Moduł umożliwiający operacje na systemie operacyjnym, takie jak zarządzanie plikami i folderami.

**pandas:** Biblioteka do analizy danych, szczególnie przydatna do pracy z danymi w formacie tabelarycznym.

**numpy:** Biblioteka do obliczeń numerycznych.

**to\_categorical z keras.utils:** Funkcja konwertująca etykiety (klasy) do formatu one-hot encoding.

**load\_img z keras\_preprocessing.image:** Funkcja do ładowania obrazów.

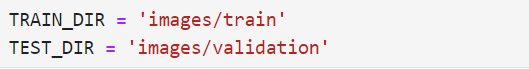
**Sequential z keras.models:** Klasa do budowy sekwencyjnego modelu sieci neuronowej.

**Dense, Conv2D, Dropout, Flatten, MaxPooling2D z keras.layers:** Warstwy używane do budowy sieci neuronowej.

**matplotlib.pyplot:** Biblioteka do tworzenia wykresów.

**%matplotlib inline:** Komenda Jupyter Notebook, która pozwala na wyświetlanie wykresów bezpośrednio w notatniku.

**Definiujemy zmienne które będą ścieżkami do katalogów zawierających dane obrazowe, które będą używane odpowiednio do treningu i testowania modelu sieci neuronowej.**

****

**TRAIN\_DIR** jest zmienną przechowującą ścieżkę do katalogu, który zawiera dane treningowe dla modelu.

**TEST\_DIR** jest zmienną przechowującą ścieżkę do katalogu, który zawiera dane walidacyjne (testowe) dla modelu.

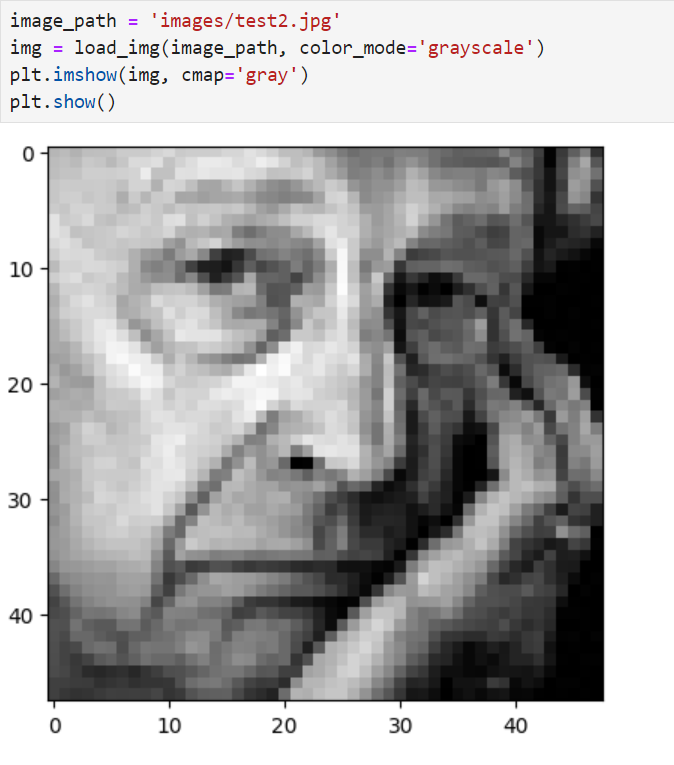
Zmienne są potrzebne do:

**Treningu modelu:** Podczas treningu modelu sieci neuronowej konieczne jest dostarczenie danych treningowych, które będą używane do nauczenia modelu rozpoznawania wzorców w danych.

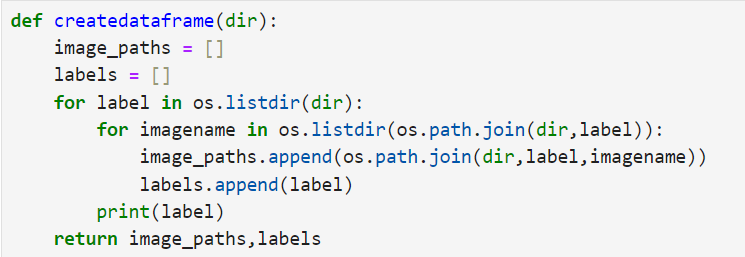
**Walidacji modelu:** Po wytrenowaniu modelu ważne jest przetestowanie jego skuteczności na niezależnym zbiorze danych, które nie były używane podczas treningu. Z tego powodu używa się oddzielnych katalogów na dane treningowe i walidacyjne.

# **3.Preprocessing danych**

**Prezentacja przykładowego obrazu w skali szarości z dataset FER-2013 na którym będziemy trenować model**

****

**Budujemy funkcje która służy do utworzenia ramy danych zawierającej ścieżki do obrazów oraz odpowiadające im etykiety**

****

**Parametr dir** jest ścieżką do katalogu, który zawiera podkatalogi (etykiety) zawierające obrazy.

**image\_paths** jest pustą listą, która będzie przechowywać pełne ścieżki do wszystkich obrazów.

**labels** jest pustą listą, która będzie przechowywać etykiety odpowiadające każdemu obrazowi.

**Pierwsza pętla for** iteruje przez zawartość katalogu głównego (os.listdir(dir)), która zawiera podkatalogi (etykiety).

for label in os.listdir(dir):

**Wewnątrz pierwszej pętli, druga pętla for** iteruje przez zawartość każdego podkatalogu, aby uzyskać listę nazw plików obrazowych (imagename).Wewnątrz tej pętli, pełna ścieżka do każdego obrazu jest tworzona za pomocą os.path.join(dir, label, imagename) i dodawana do listy image\_paths.

**Etykieta** (nazwa podkatalogu) jest dodawana do listy labels za pomocą labels.append(label).

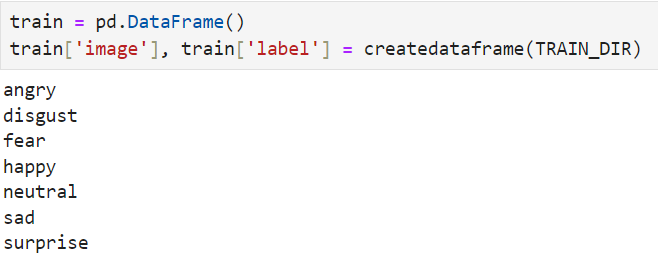
labels.append(label)

**print(label)** wyświetla nazwę aktualnej etykiety na konsoli. To może być użyteczne do monitorowania postępu procesu tworzenia ramy danych.

**Funkcja zwraca dwie wartości:** image\_paths (lista ścieżek do obrazów) i labels (lista odpowiadających im etykiet).

**Ostatecznie ta funkcja umożliwi łatwe zbieranie danych obrazowych z podanego katalogu, przetwarzając etykiety i ścieżki do plików, jest to przygotowanie danych przed ich załadowanie do modelu sieci neuronowej.**

**Tworzymy ramę danych którą wypełnimy ją danymi uzyskanymi z funkcji createdataframe(dir)**

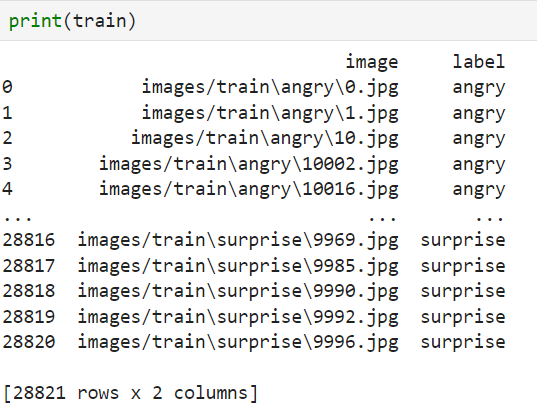
****

**train['image']** przypisuje listę image\_paths do kolumny 'image' w ramie danych train.

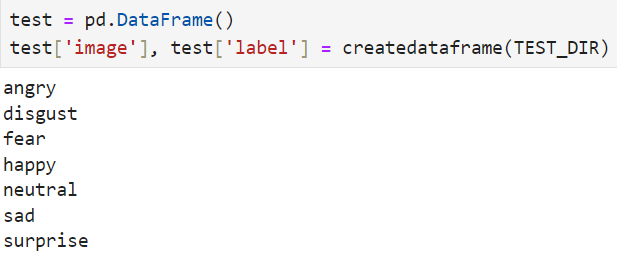
**train['label']** przypisuje listę labels do kolumny 'label' w ramie danych train.

**Ten kod** wypełni ramę danych train danymi, które reprezentują obrazy i ich etykiety.

**Wydruk pełną zawartość ramy danych “train”**

****

**Tworzymy kolejną ramę danych którą wypełnimy ją danymi uzyskanymi z funkcji createdataframe(dir)**

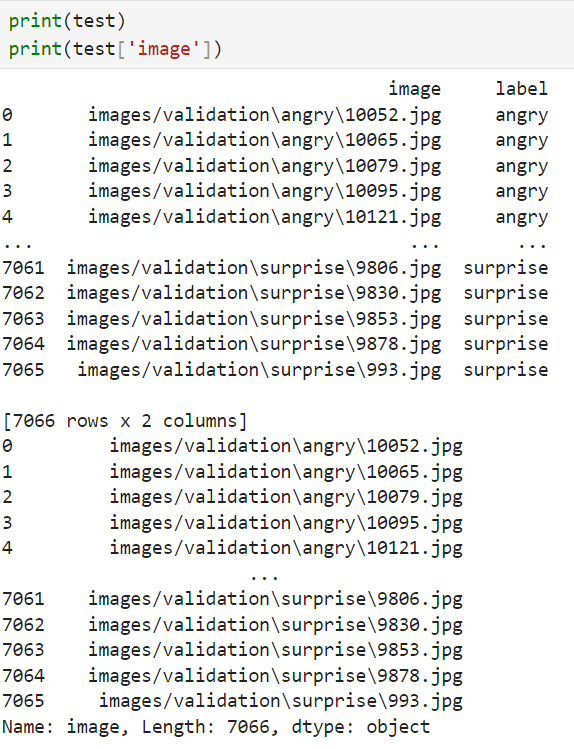
****

**createdataframe(TEST\_DIR)** jest funkcją, która zwraca dwie wartości: image\_paths (lista ścieżek do obrazów) i labels (lista odpowiadających im etykiet).

**test['image']** przypisuje listę image\_paths do kolumny 'image' w ramie danych test.

**test['label']** przypisuje listę labels do kolumny 'label' w ramie danych test.

**Wydruk ramy danych danych oraz kolumn z tej ramy danych**

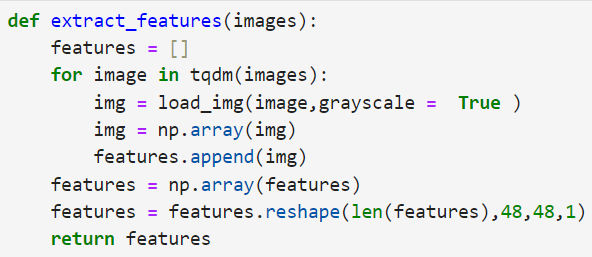
****

**Przetwarzamy obrazy na cechy**

**Importujemy tqdm** które jest używane do wyświetlania paska postępu, co jest przydatne podczas przetwarzania dużej liczby obrazów.

****

**Tworzymy funkcję extract\_features(images)** która będzie przetwarzała obrazy na cechy, które będą użyte jako dane wejściowe do modelu uczenia maszynowego

****

**images**: Jest to lista ścieżek do obrazów, które mają być przetworzone na cechy.

**features = []** Tworzy pustą listę features, która będzie przechowywać przetworzone cechy obrazów.

**for image in tqdm(images):** Pętla for z użyciem tqdm która

iteruje przez listę images przy użyciu tqdm do wyświetlania paska postępu.

**load\_img(image, grayscale=True):** Funkcja load\_img z biblioteki Keras ładuje obraz o ścieżce image i konwertuje go na obraz w skali szarości (grayscale=True).

***Używamy konwersji*** *do skali szarości ponieważ jest bardzo przydatna w rozpoznawaniu twarzy, powoduje to że kolor przestaje być kluczowy, a istotny jest jedynie kształt i struktura obiektów.*

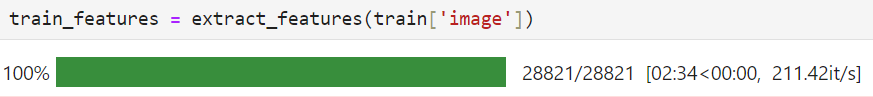
**np.array(img)**: Konwertuje załadowany obraz na tablicę numpy.

**features.append(img):** Dodaje przetworzony obraz do listy cech features.

**np.array(features):** Konwertujesz listę cech features na tablicę numpy.

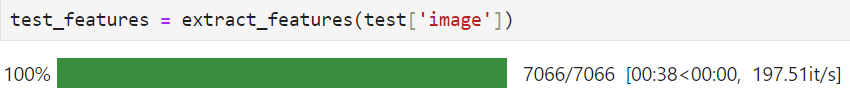
**features.reshape(len(features), 48, 48, 1):** Zmieniasz kształt tablicy cech, aby odpowiednio odpowiadał wymaganiom wejściowym modelu. W tym przypadku, obrazy są przekształcane do wymiaru (48, 48, 1), gdzie 1 oznacza kanał (skala szarości).

**Przypisujemy wyniki funkcji do zmiennych które będą używane jako dane wejściowe do modelu uczenia maszynowego**

****

**train['image']** jest kolumną ramy danych train, która zawiera ścieżki do obrazów treningowych.

**extract\_features(train['image'])** wywołuje funkcję extract\_features, przekazując jako argument listę ścieżek do obrazów treningowych.

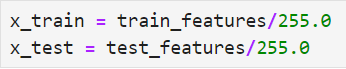
****

**test['image']** jest kolumną ramy danych test, która zawiera ścieżki do obrazów testowych.

**extract\_features(test['image'])** wywołuje funkcję extract\_features, przekazując jako argument listę ścieżek do obrazów testowych.

**Funkcja extract\_features** W obu przyadkach przetwarza każdy obraz testowy (za pomocą wczytania, przekształcenia na skali szarości i konwersji na tablicę numpy) i zwraca je jako tablicę test\_features.

**Normalizacja cech**

****

**Normalizacja danych** jest ważnym krokiem w przetwarzaniu danych obrazowych, szczególnie gdy wartości pikseli są wyrażane w przedziale od 0 do 255 (np. w przypadku obrazów w skali szarości). Normalizacja ma na celu sprowadzenie wartości cech do przedziału 0-1 lub do średniej zero i wariancji jednostkowej, co może przyspieszyć uczenie modelu i poprawić jego stabilność.

**Jak działa normalizacja?**

**Dzielenie przez 255.0:** W przypadku danych obrazowych, gdzie każdy piksel ma wartość od 0 do 255 (jeśli jest to skala szarości), dzieląc przez 255.0, otrzymujemy wartości pikseli z zakresu 0.0 do 1.0. Jest to realizowane za pomocą operacji element-wise na tablicy cech (train\_features i test\_features), gdzie każdy piksel w każdym obrazie jest dzielony przez 255.0.

**Jakie możemy mieć korzyści z tej normalizacji?**

**Uczenie szybsze i stabilniejsze:** Model może szybciej znajdować optymalne rozwiązania, gdy dane są w mniejszym zakresie.

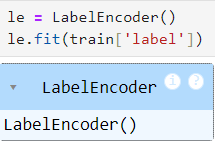
**Unikanie problemów z optymalizacją:** Może to pomóc uniknąć problemów z gradientem, które mogą wystąpić, gdy wartości wejściowe są bardzo duże.

**Kodowanie etykiet**

**Zaczynamy od Importowania LabelEncoder**, który jest używany do kodowania etykiet kategorialnych w formie liczb.



**Następnie tworzymy obiekt LabelEncoder** i przypisujesz go do zmiennej le. LabelEncoder przekształca etykiety kategorialne na liczby całkowite.

****

**train['label']** to kolumna w ramie danych train, która zawiera etykiety (np. nazwy klas).

**le.fit(train['label'])** dopasowuje LabelEncoder do unikalnych wartości w kolumnie train['label'], czyli uczy się mapowania każdej unikalnej etykiety na odpowiadającą jej liczbę całkowitą.

**Działanie tego LabelEncoder’a** polega na tym że przypisuje on każdej unikalnej wartości w kolumnie train['label'] unikalną liczbę całkowitą, zaczynając od 0.

Na przykład, jeśli masz etykiety 'Smutek', 'Gniew', 'Neutralny', LabelEncoder może przypisać im odpowiednio liczby 0, 1, 2.

**Po wykonaniu le.fit(train['label'])**, LabelEncoder przechowuje mapowanie etykiet na liczby całkowite w swojej wewnętrznej pamięci. Te liczby całkowite będą później używane do przekształcania etykiet treningowych i testowych na postać numeryczną, które będą używane przez model.

**Przekształcenie etykiet kategorialnych (nazw klas) na liczby całkowite za pomocą wcześniej zainicjowanego obiektu kodowania etykiet**



**le.transform(train['label'])** przekształca etykiety w kolumnie

train['label'] na odpowiadające im liczby całkowite zgodnie z wcześniej ustalonym mapowaniem LabelEncoder.

Wynikowe liczby całkowite są przypisywane do y\_train.

**Podobnie, le.transform(test['label'])** przekształca etykiety w kolumnie test['label'] na liczby całkowite zgodnie z tym samym mapowaniem LabelEncoder.

Wynikowe liczby całkowite są przypisywane do y\_test.

Przekształcamy etykiety ponieważ model i sieci neuronowe wymagają etykiety klas w postaci liczbowej, a nie tekstowej, w naszym wypadku to **“LabelEncoder”** zapewnia prosty sposób na przekształcenie etykiet kategorialnych na liczby całkowite, co umożliwia modelom pracy z tymi danymi

**Przekształcenie zakodowanych etykiet klas na formę one-hot**

W wyniku tego że będziemy stosowali używali funkcji **softmax**(która opiszemy w dalszej części dokumentacji) jako funkcji aktywacji w warstwie wejściowej musimy zaprezentować etykiety klas w postaci one-hot encoding



**y\_train** jest tablicą zawierającą numeryczne etykiety klas, które zostały wcześniej zakodowane za pomocą LabelEncoder.

**to\_categorical(y\_train, num\_classes=7)** przekształca te numeryczne etykiety klas na formę one-hot encoding. Oznacza to, że każda liczba całkowita zostaje zamieniona na wektor binarny (bitowy) o długości równej liczbie klas (num\_classes=7 w tym

**Podobnie jak dla y\_train, y\_test**, które zawiera numeryczne etykiety klas dla danych testowych, jest przekształcane w formę one-hot encoding za pomocą to\_categorical.przypadku).

Dział to w taki sposób że zawierający numeryczne numeryczne etykiety klas [0, 1, 2, 1, 0, 2] “y\_train”, to po to\_categorical(y\_test,num\_classes = 7) otrzymamy np:

[[[0. 1. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

...

[1. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]]

[[0. 1. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

...

[1. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]]  **. . . itd.**

Dzięki temu każda etykieta klas jest reprezentowana jako wektor binarny, co jest wymagane przez wiele modeli uczenia maszynowego.

# **4.Budowanie modelu konwolucyjnej sieci neuronowej**

****

**Definicja modelu sekwencyjnego**

****

Tworzymy pusty model sekwencyjny, do którego będziemy dodawać warstwy. Model sekwencyjny umożliwia budowanie sieci neuronowej warstwa po warstwie, co jest prostym i intuicyjnym sposobem definiowania modeli głębokiego uczenia.

**Pierwsza warstwa konolucyjna**

****

Dodajemy warstwę konwolucyjna.Warstwy konwolucyjne są kluczowe dla CNN, ponieważ mogą automatycznie wyodrębniać cechy z obrazów, takie jak krawędzie, tekstury i inne ważne struktury.

**Conv2D(128, kernel\_size=(3,3), activation='relu', input\_shape=(48,48,1)):** 128: Liczba filtrów w warstwie konwolucyjnej. Więcej filtrów pozwala na wykrywanie większej liczby cech.

**kernel\_size=(3,3):** Rozmiar filtrów (3x3). Mniejsze filtry są w stanie wyłapywać drobniejsze szczegóły w obrazie.

**activation='relu':** Funkcja aktywacji ReLU (Rectified Linear Unit), która wprowadza nieliniowość do modelu i pomaga uniknąć problemów z gradientem zanikania.

**input\_shape=(48,48,1):** Kształt danych wejściowych (48x48 pikseli, 1 kanał - skala szarości).

**Warstwa max-pooling i dropout**

****

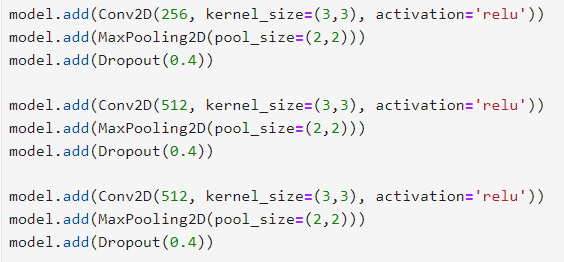
**MaxPooling2D:** Redukcja wymiarów danych wejściowych, o prowadzi do redukcji liczby parametrów w sieci i zmniejszenia obciążenia obliczeniowego. Max pooling działa na zasadzie przeglądania sąsiednich wartości w macierzy i wybierania największej wartości z tego regionu.

**Dropout:** Regularizacja modelu, losowo wyłącza (ustawia na zero) pewien procent neuronów w trakcie treningu. Pomaga to zapobiegać przeuczeniu (overfitting), gdzie model zbyt dobrze uczy się danych treningowych, ale nie generalizuje dobrze na nowe, niewidziane dane.

**MaxPooling2D(pool\_size=(2,2)):** Warstwa max-pooling zmniejsza wymiary danych wejściowych, redukując ilość obliczeń oraz chroniąc przed przeuczeniem. Wybiera maksymalną wartość z okna o rozmiarze 2x2.

**Dropout(0.4):** Losowo wyłącza 40% neuronów podczas treningu, aby zapobiec przeuczeniu (overfitting).

**Kolejne warstwy konwolucyjne, max-pooling i dropout**

****

Dodajemy kolejne warstwy konwolucyjne, max-pooling i dropout.

Zwiększanie liczby filtrów w kolejnych warstwach konwolucyjnych (256, 512, 512) pozwala na wyodrębnianie bardziej złożonych cech.

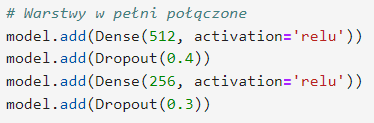
Kolejne warstwy max-pooling kontynuują redukcję wymiarów danych. Warstwy dropout wciąż pomagają zapobiegać przeuczeniu.

**Warstwa spłaszczająca (Flatten)**

****

Następuje tutaj przekształcenie danych wejściowych do jednowymiarowego wektora. Przed przejściem do warstw w pełni połączonych (dense), dane muszą być spłaszczone, aby mogły być wprowadzone do tych warstw.

**Warstwy w pełni połączone (Dense) i dropout**

****

Dodajemy warstwy w pełni połączone i warstwy dropout.

**Dense(512, activation='relu'):** Warstwa w pełni połączona z 512 neuronami, która umożliwia modelowi łączenie cech wyodrębnionych przez warstwy konwolucyjne.

**Dropout(0.4):** Dropout wyłącza 40% neuronów, aby zapobiec przeuczeniu.

**Dense(256, activation='relu'):** Druga warstwa w pełni połączona z 256 neuronami, dalej przetwarzająca cechy.

**Dropout(0.3):** Dropout wyłącza 30% neuronów, aby zapobiec przeuczeniu.

****

Dodajemy warstwę wyjściową z 7 neuronami.

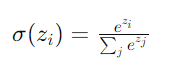
Dense(7, activation='softmax'): Warstwa wyjściowa z 7 neuronami (odpowiadająca 7 klasom emocji) i funkcją aktywacji softmax, która zwraca rozkład prawdopodobieństwa dla każdej klasy. Funkcja softmax przekształca wyjścia w prawdopodobieństwa, które sumują się do 1.

**Dlaczego używamy akurat softmax?**

Funkcja aktywacji softmax jest szczególnie użyteczna w przypadku problemów klasyfikacyjnych, w których celem jest przypisanie obserwacji do jednej z wielu klas. W przypadku modelu do wykrywania emocji, softmax jest używany w warstwie wyjściowej z kilku powodów:

**1. Wynik w postaci prawdopodobieństw**

Softmax przekształca surowe wyniki (logity) wyjściowe modelu w wartości prawdopodobieństwa. Dla każdej klasy iii, wartość wyjściowa **“i”**​ jest przekształcana za pomocą funkcji:



To oznacza, że suma wszystkich wartości wyjściowych dla poszczególnych klas wynosi 1, co pozwala interpretować wyniki jako prawdopodobieństwa.

### **2. Jednoznaczna klasyfikacja**

Ponieważ softmax daje prawdopodobieństwa, możemy łatwo wybrać klasę z najwyższym prawdopodobieństwem jako przewidywaną klasę. To jest kluczowe w problemach, gdzie każda obserwacja należy do dokładnie jednej klasy, takich jak wykrywanie emocji.

### **3. Stabilność numeryczna**

Funkcja softmax ma właściwości, które pomagają w stabilności numerycznej podczas treningu modelu. Operując na logitach i ekspresjach wykładniczych, softmax pomaga w unikaniu problemów związanych z dużymi różnicami w wartościach logitów.

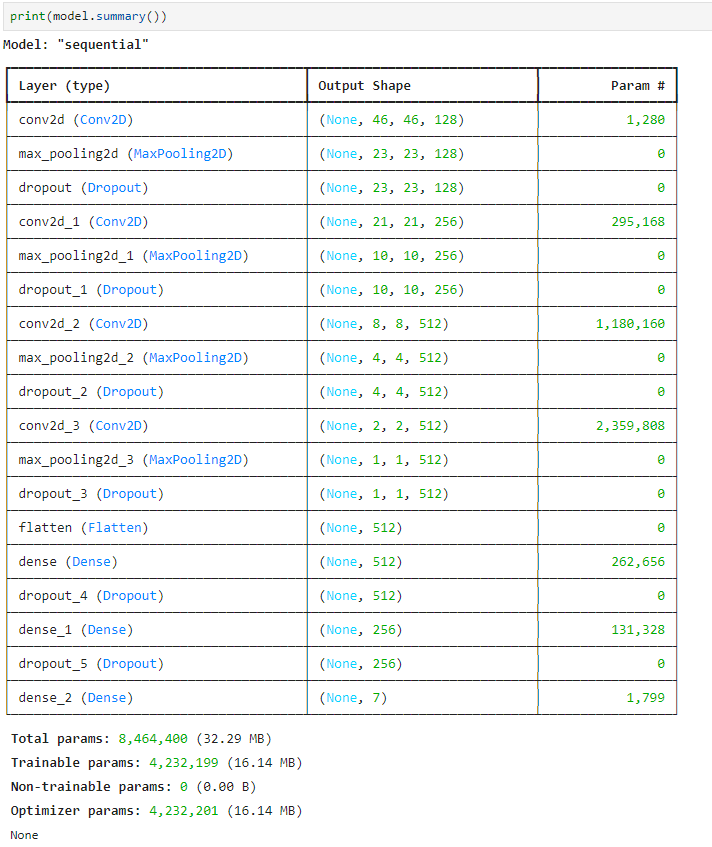
### **4. Gradienty podczas backpropagacji**

Softmax dobrze współpracuje z funkcją kosztu cross-entropy (entropia krzyżowa), która jest często używana w klasyfikacji wieloklasowej. Połączenie softmax i cross-entropy sprawia, że gradienty są dobrze zdefiniowane i efektywne do trenowania modelu za pomocą backpropagacji.

### **5. Wymagania problemu wieloklasowego**

W problemie wykrywania emocji mamy więcej niż dwie klasy (np. Gniew, Obrzydzenie, Strach, Wesoły, Neutralny, Smutek, Zaskoczenie). Softmax jest zaprojektowany do obsługi wielu klas jednocześnie, co czyni go idealnym wyborem dla takich problemów.

**Podsumowanie Architektury sieci neuronowej**

****

**Podsumowanie naszego modelu**

Model ten jest przykładem głębokiej sieci konwolucyjnej (CNN) stosowanej do klasyfikacji obrazów. Kolejne warstwy konwolucyjne wyodrębniają cechy, warstwy max-pooling redukują wymiary danych, a warstwy dropout zapobiegają przeuczeniu. Warstwy w pełni połączone przetwarzają wyodrębnione cechy, a warstwa wyjściowa z funkcją softmax zwraca prawdopodobieństwa dla każdej z klas emocji. Model jest gotowy do kompilacji i trenowania na danych treningowych z data set’u FER-2013**.**

**Przygotowywanie modelu do treningu**

Kompilacja modelu to etap, na którym określamy jak model ma być trenowany. Proces ten obejmuje wybór optymalizatora, funkcji straty oraz metryk oceny. W tym etapie określamy następująco:

**Konfiguracje optymalizatora**: Ustawienia optymalizatora (np.

tempo uczenia, beta1, beta2 w przypadku Adama) są zainicjowane.

**Definiowanie funkcji straty**: Określana jest funkcja straty, która będzie używana do obliczania, jak dobrze model radzi sobie z przewidywaniem etykiet.

**Określanie metryk**: Metryki, które będą używane do monitorowania wydajności modelu podczas treningu i testowania, są zdefiniowane.

****

**optimizer='adam':** W naszym wypadku optymalizator który wybraliśmy to **Adam (Adaptive Moment Estimation):** jest to popularny optymalizator w uczeniu maszynowym, który łączy zalety dwóch innych optymalizatorów: AdaGrad (który dobrze radzi sobie z rzadkimi gradientami) i RMSProp (który dobrze radzi sobie z gradientami, które są bardzo różne w czasie).

Używa dwóch momentów (średnich ruchomych pierwszego i drugiego rzędu) do adaptacyjnego dostosowania tempa uczenia się każdego parametru. Adam jest preferowany ze względu na swoją efektywność i stosunkowo dobrą wydajność w różnych problemach uczenia się.

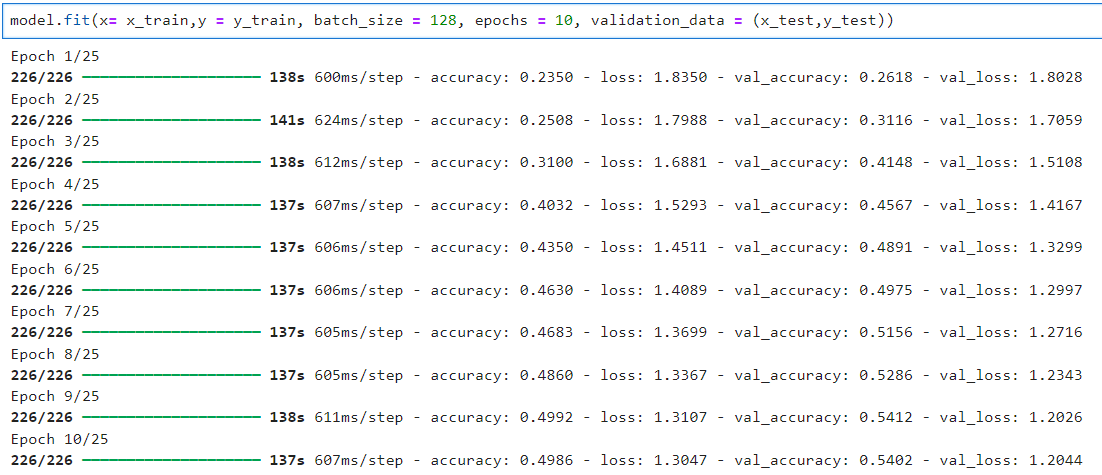
**loss='categorical\_crossentropy':** wybraliśmy Kategorie krzyżową jest to funkcja straty używana w problemach klasyfikacji wieloklasowej, gdzie klasy są zakodowane w formacie one-hot.

Funkcja ta mierzy różnicę (dywergencję Kullbacka-Leiblera) między prawdziwym rozkładem etykiet a przewidywanym rozkładem przez model. Minimalizacja tej straty prowadzi do poprawy dokładności klasyfikacji modelu.

**metrics=['accuracy']: jako metryka oceny w zadaniach klasyfikacji wybraliśmy** dokładność(accuracy) jest to miara, która wskazuje, jak dobrze model przewiduje prawidłowe etykiety.

Dokładność jest definiowana jako stosunek liczby poprawnie przewidzianych etykiet do ogólnej liczby etykiet.

# **5.Trening i testowanie stworzonego modelu**

****

**x=x\_train(Dane treningowe):** w tym przypadku, x\_train to zestaw cech, które zostały wstępnie przetworzone i znormalizowane (wartości między 0 a 1).

**y=y\_train**: etykiety treningowe y\_train to “one-hot encoded” etykiety dla danych treningowych.

**batch\_size=128:** liczba próbek które zostaną przetworzone przed zaktualizowaniem modelu. W tym przypadku, 128 próbek jest przetwarzanych na raz.

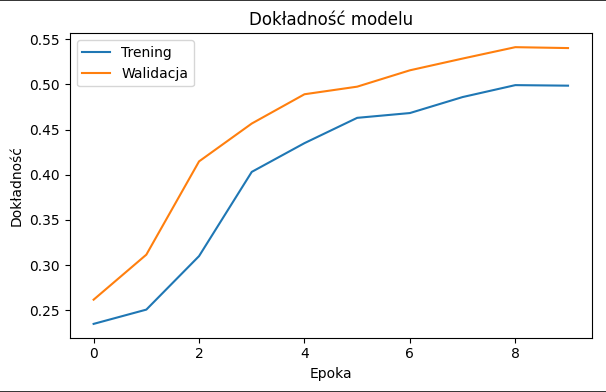
**epochs=10:** liczba epok przez które model będzie trenowany. Jedna epoka oznacza przejście przez cały zestaw danych treningowych.

**validation\_data=(x\_test, y\_test):** dane walidacyjne, te dane są używane do oceny wydajności modelu po każdej epoce. x\_test to zestaw cech walidacyjnych, a y\_test to “one-hot encoded” etykiety walidacyjne.

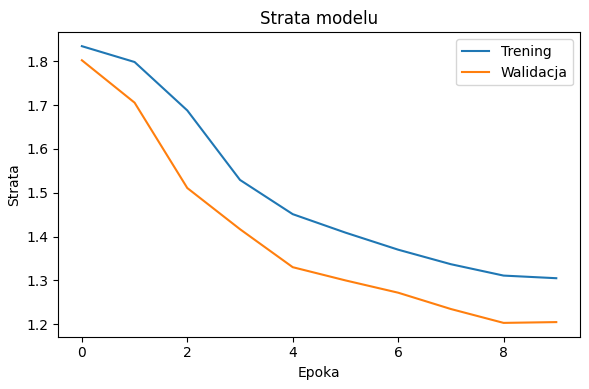
**Analiza wyników trenowania modelu**

****

**Wykres dokładności modelu**

****

**Wykres starty modelu**

****

**Wnioski odnośnie wyników treningu**

**Na podstawie wyników trenowania modelu przez 10 epok, można zauważyć pewne istotne obserwacje:**

**Dokładność na zbiorze treningowym:** Początkowo dokładność rosła z 23.5% w pierwszej epoce do około 49.9% w dziewiątej epoce, a następnie stabilizowała się na poziomie około 49.9% w dziesiątej epoce.

**Dokładność na zbiorze walidacyjnym:** Podobnie jak w przypadku zbioru treningowego, dokładność na zbiorze walidacyjnym wzrosła z 26.2% w pierwszej epoce do około 54.0% w dziesiątej epoce. To sugeruje, że model miał zdolność generalizacji, czyli radził sobie również z nowymi danymi, które nie były używane podczas treningu.

**Strata (loss):** Strata zmniejszała się w miarę postępu treningu, co jest oczekiwanym zachowaniem. Niższa strata wskazuje na to, że model uczył się lepiej dostosowywać do danych treningowych i walidacyjnych.

**Ocena ogólna:** Wyniki te wskazują, że model posiada zdolność do nauki i generalizacji, ale dokładność na poziomie około 50% nie jest zbyt wysoka. Można by spróbować poprawić wyniki poprzez dostosowanie architektury modelu (np. zmiana liczby warstw i ich rozmiarów), dostrojenie hiperparametrów (np. szybkość uczenia, współczynniki dropout), zwiększenie zbioru treningowego lub ulepszenie technik augmentacji danych.

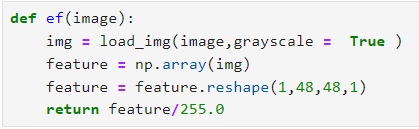
**Kontynuacja optymalizacji:** Aby uzyskać lepsze wyniki, warto rozważyć dalszą optymalizację modelu poprzez eksperymentowanie z różnymi architekturami sieci, hiperparametrami i technikami regularyzacji. Można również rozważyć użycie pre-trenowanych modeli, jeśli dostępne są modele przeszkolone na dużych zbiorach danych podobnych do tego problemu.

**Test zdolności do wykrywania emocji naszego modelu**

Przygotowywujemy środowisko do prostego testowania działania naszego modelu



Przyporządkowujemy etykiety do odpowiadających im emocji

****

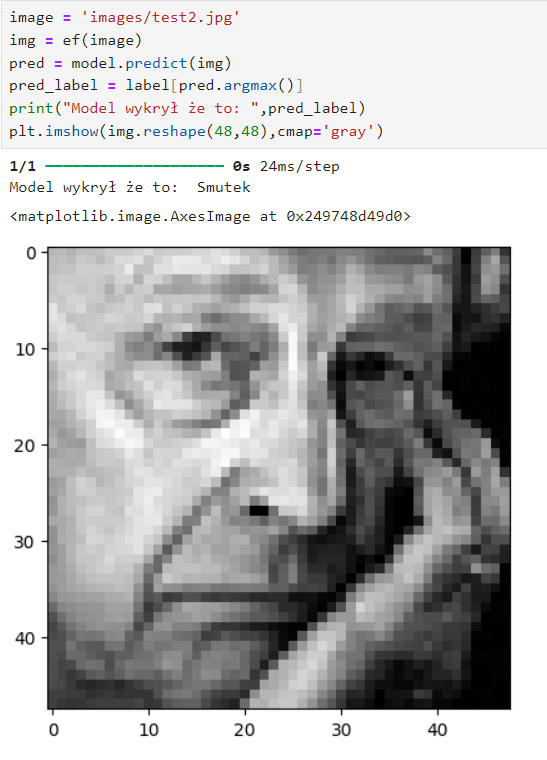
**Funkcja ef(image)** przetwarza pojedynczy obraz w skali szarości o rozmiarze 48x48 pikseli na odpowiedni format, który można przekazać do modelu do predykcji. Normalizacja przez dzielenie przez 255 pomaga uniknąć problemów z gradientem i przyspiesza uczenie się modelu poprzez skalowanie wartości pikseli do zakresu od 0 do 1.

**Teraz możemy przetestować nasz model na podstawie obrazów z naszej bazy obrazów testowych**









# **6.Implementacja prostego interfejsu użytkownika obsługująca model wykrywania emocji**

Interfejs ten umożliwia użytkownikowi w czasie rzeczywistym obserwowanie przewidywanych emocji na twarzach widocznych na ekranie za pomocą kamery wideo.

**# -\*- coding: utf-8 -\*-**

**import cv2**

**from tensorflow.keras.models import model\_from\_json**

**import numpy as np**

**import tkinter as tk**

**from tkinter import ttk**

**from PIL import Image, ImageTk**

**import time**

**# Ścieżki do plików modelu**

**json\_file\_path = "C:/Users/micha/Desktop/Wykrywanie\_emocji/model50p.json"**

**weights\_file\_path = "C:/Users/micha/Desktop/Wykrywanie\_emocji/model50p.h5"**

**# Wczytanie architektury modelu z pliku JSON**

**with open(json\_file\_path, "r", encoding="utf-8") as json\_file:**

**model\_json = json\_file.read()**

**# Stworzenie modelu na podstawie architektury z pliku JSON**

**model = model\_from\_json(model\_json)**

**# Wczytanie wytrenowanych wag do modelu**

**model.load\_weights(weights\_file\_path)**

**# Ścieżka do pliku haarcascade**

**haar\_file = cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml'**

**# Utworzenie klasyfikatora kaskadowego do detekcji twarzy**

**face\_cascade = cv2.CascadeClassifier(haar\_file)**

**# Funkcja do przetwarzania obrazu**

**def extract\_features(image):**

**feature = np.array(image)**

**feature = feature.reshape(1, 48, 48, 1)**

**return feature / 255.0**

**# Inicjalizacja źródła wideo**

**webcam = None**

**# Słownik etykiet emocji**

**labels = {0: 'Gniew', 1: 'Obrzydzenie', 2: 'Strach', 3: 'Wesoly', 4: 'Neutralny', 5: 'Smutek', 6: 'Zaskoczenie'}**

**# Zmienna do przechowywania czasu ostatniej predykcji**

**last\_prediction\_time = 0**

**# Funkcja do aktualizacji klatek wideo w Tkinter**

**def update\_frame():**

**global last\_prediction\_time**

**ret, frame = webcam.read()**

**if not ret:**

**print("Błąd odczytu strumienia")**

**return**

**# Przetwarzanie klatki: konwersja na odcień szarości i detekcja twarzy**

**gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 5)**

**current\_time = time.time()**

**interval = freq\_slider.get()**

**if current\_time - last\_prediction\_time >= interval:**

**try:**

**for (x, y, w, h) in faces:**

**face\_image = gray[y:y+h, x:x+w]**

**cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)**

**face\_image = cv2.resize(face\_image, (48, 48))**

**img = extract\_features(face\_image)**

**pred = model.predict(img)**

**prediction\_label = labels[pred.argmax()]**

**cv2.putText(frame, '%s' % prediction\_label, (x-10, y-10), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 2, (255, 0, 0))**

**last\_prediction\_time = current\_time**

**except cv2.error as e:**

**print(f"Błąd związany z CV2: {e}")**

**# Konwersja obrazu do formatu kompatybilnego z Tkinter**

**cv2image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGBA)**

**img = Image.fromarray(cv2image)**

**imgtk = ImageTk.PhotoImage(image=img)**

**lmain.imgtk = imgtk**

**lmain.configure(image=imgtk)**

**# Ustawienie częstotliwości aktualizacji klatek**

**lmain.after(10, update\_frame) # Aktualizacja co 10 ms (około 100 FPS)**

**# Funkcja do rozpoczęcia odczytu z wybranego źródła**

**def start\_stream():**

**global webcam**

**source = source\_var.get()**

**if source == "Kamerka lokalna":**

**webcam = cv2.VideoCapture(0)**

**elif source == "DroidCam":**

**address = 'http://adres:port/video' # Przykładowy adres IP i port**

**webcam = cv2.VideoCapture(address)**

**if not webcam.isOpened():**

**print("Nie można otworzyć strumienia wideo")**

**return**

**print(f"Strumień wideo został pomyślnie otwarty")**

**update\_frame()**

**# Tworzenie aplikacji Tkinter**

**root = tk.Tk()**

**root.title("Wykrywanie Emocji")**

**# Ustawienie minimalnych wymiarów okna**

**root.minsize(width=640, height=480)**

**# Ustawienie skalowalności okna**

**root.resizable(width=True, height=True)**

**# Wczytanie obrazu na tło aplikacji**

**background\_image = Image.open("C:/Users/micha/Desktop/Wykrywanie\_emocji/BG.webp")**

**background\_image = background\_image.resize((640, 480), Image.LANCZOS) # Dopasowanie rozmiaru do okna**

**background\_photo = ImageTk.PhotoImage(background\_image)**

**# Ustawienie tła dla okna głównego**

**background\_label = tk.Label(root, image=background\_photo)**

**background\_label.place(x=0, y=0, relwidth=1, relheight=1)**

**# Wyświetlanie wideo**

**lmain = tk.Label(root)**

**lmain.pack()**

**# Etykieta dla suwaka**

**freq\_label = tk.Label(root, text="Częściej Częstotliwość wykrywania (sekundy) Rzadziej")**

**freq\_label.pack()**

**# Dodanie suwaka do kontrolowania częstotliwości przetwarzania klatek**

**freq\_slider = ttk.Scale(root, from\_=0.1, to=5, orient='horizontal', length=200)**

**freq\_slider.set(1) # Ustawienie domyślnej wartości na 1 sekunda**

**freq\_slider.pack()**

**# Dodanie opcji wyboru źródła wideo**

**source\_label = tk.Label(root, text="Wybierz źródło wideo")**

**source\_label.pack()**

**source\_var = tk.StringVar(value="Kamerka lokalna")**

**source\_options = ["Kamerka lokalna", "DroidCam"]**

**source\_menu = ttk.Combobox(root, textvariable=source\_var, values=source\_options)**

**source\_menu.pack()**

**# Dodanie przycisku do rozpoczęcia strumienia**

**start\_button = tk.Button(root, text="Start/Stop", command=start\_stream)**

**start\_button.pack()**

**# Uruchomienie aplikacji**

**root.mainloop()**

**# Zwolnienie obiektu VideoCapture i zamknięcie okien**

**if webcam:**

**webcam.release()**

**cv2.destroyAllWindows()**

**Kluczowe elementy tego Interfejsu**

**Wczytywanie modelu:** Interfejs wczytuje wytrenowany model sieci neuronowej z pliku JSON opisującego architekturę oraz z pliku H5 zawierającego wytrenowane wagi.

**Detekcja twarzy:** Wykorzystuje klasyfikator kaskadowy HaarCascade do detekcji twarzy na obrazie wideo. Twarze są wyróżniane na ekranie za pomocą prostokątów.

**Przetwarzanie obrazu:** Każda wykryta twarz jest konwertowana do odcieni szarości, zmniejszana do rozmiaru 48x48 pikseli i normalizowana przed przekazaniem do modelu do przewidywania emocji.

**Wyświetlanie wyników:** Na obrazie wideo wyświetlane są przewidywane emocje w postaci tekstowej nad każdą wykrytą twarzą.

**Funkcje interfejsu:**

-Wyświetlane jest wideo w interfejsie Tkinter za pomocą Label o nazwie lmain.

-Dostępny jest suwak do regulacji częstotliwości aktualizacji klatek wideo.

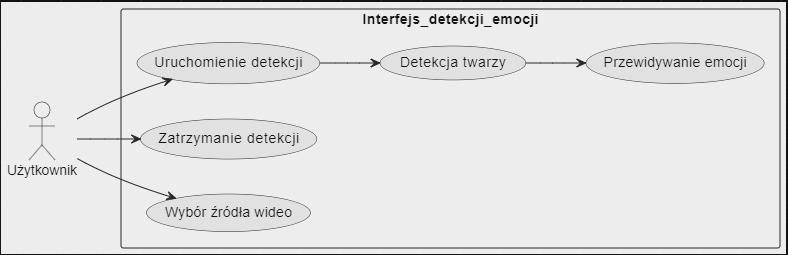
-Użytkownik może wybrać źródło wideo między lokalną kamerą a DroidCam.

-Przycisk "Start/Stop" uruchamia lub zatrzymuje strumień wideo.

**Zarządzanie czasem:** Aplikacja ogranicza częstotliwość przewidywania emocji za pomocą suwaka, aby zwiększyć efektywność przetwarzania.

**Zakończenie działania:** Po zakończeniu działania aplikacji zasoby kamery są zwalniane, a wszystkie otwarte okna są zamykane.

**Diagram Use Case tego interfejsu**

****

**Objaśnienie diagramu przypadków użycia:**

**Użytkownik:** Osoba korzystająca z interfejsu detekcji emocji.

Interfejs\_detekcji\_emocji: System, który obsługuje detekcję emocji.

Uruchomienie detekcji: Użytkownik może rozpocząć proces detekcji emocji na obrazie z kamery.

**Zatrzymanie detekcji:** Użytkownik może zakończyć proces detekcji emocji.

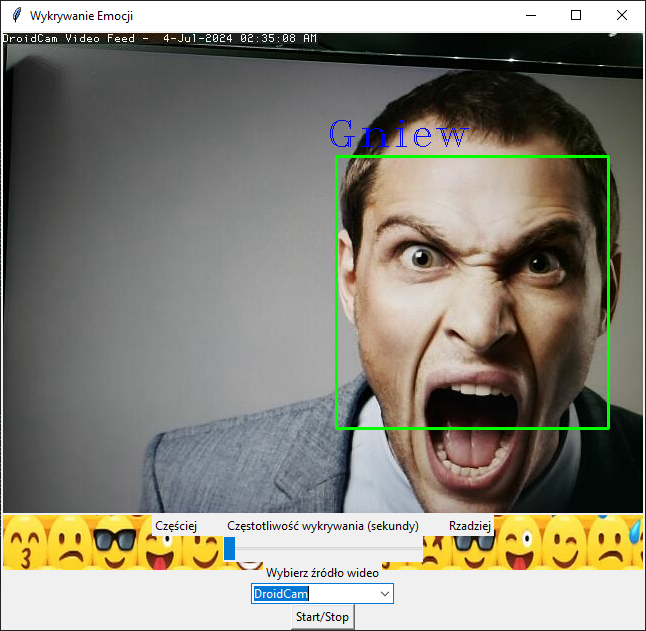
**Wybór źródła wideo:** Użytkownik może wybrać, czy detekcja ma odbywać się z kamery lokalnej czy przez DroidCam.

**Detekcja twarzy:** Etap procesu detekcji emocji, w którym system wykrywa twarze na obrazie.

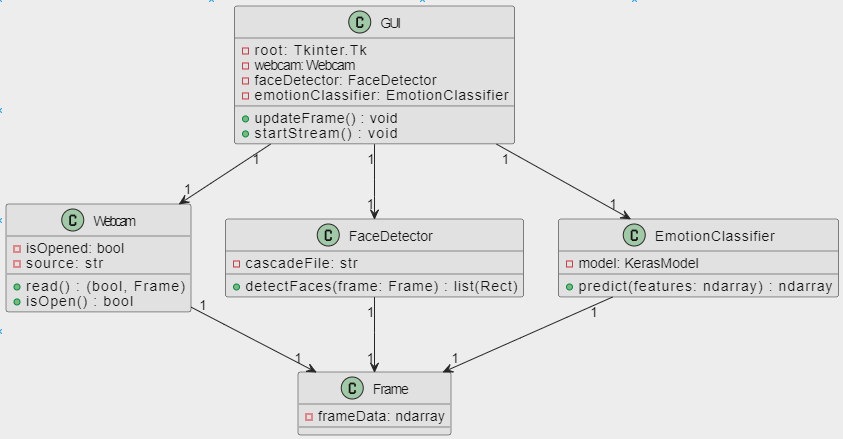
**Przewidywanie emocji:** Etap procesu detekcji emocji, w którym system analizuje wykryte twarze i określa przypisane emocje.

**Wizualizacja działania interfejsu**

Pokaz działania interfejsu w czasie wykrywania emocji z losowego obrazka z internetu



**Diagram klas naszego projektu**

****

**Objasnienie diagramu klas:**

**Webcam:** Klasa odpowiedzialna za obsługę kamery wideo. Ma metody do odczytu klatek wideo (read()) i sprawdzenia stanu połączenia (isOpened()).

**FaceDetector:** Klasa do detekcji twarzy przy użyciu klasyfikatora kaskadowego. Metoda detectFaces() zwraca listę prostokątów (Bounding Box) zawierających wykryte twarze na klatce wideo.

**EmotionClassifier:** Klasa, która obsługuje model do klasyfikacji emocji. Metoda predict() przyjmuje cechy obrazu (w naszym przypadku przetworzone fragmenty twarzy) i zwraca przewidywane emocje.

**GUI:** Klasa reprezentująca interfejs użytkownika oparty na Tkinter. Zawiera metody do aktualizacji klatek wideo (updateFrame()) i rozpoczęcia strumienia (startStream()). Posiada również referencje do obiektów Webcam, FaceDetector i EmotionClassifier.

Frame: Klasa reprezentująca pojedynczą klatkę wideo jako macierz numpy (ndarray). Jest używana jako klasa pomocnicza przez Webcam, FaceDetector i EmotionClassifier.

**Relacje:**

Webcam ma kompozycyjny związek z Frame, ponieważ każda klatka wideo jest częścią kamery.

FaceDetector i EmotionClassifier również mają związki kompozycyjne z Frame, ponieważ operują na danych klatki wideo.

GUI posiada związki asocjacyjne z Webcam, FaceDetector i EmotionClassifier, ponieważ używa tych obiektów do obsługi strumienia wideo i analizy emocji.