Konwolucyjne sieci neuronowe w klasyfikacji emocji

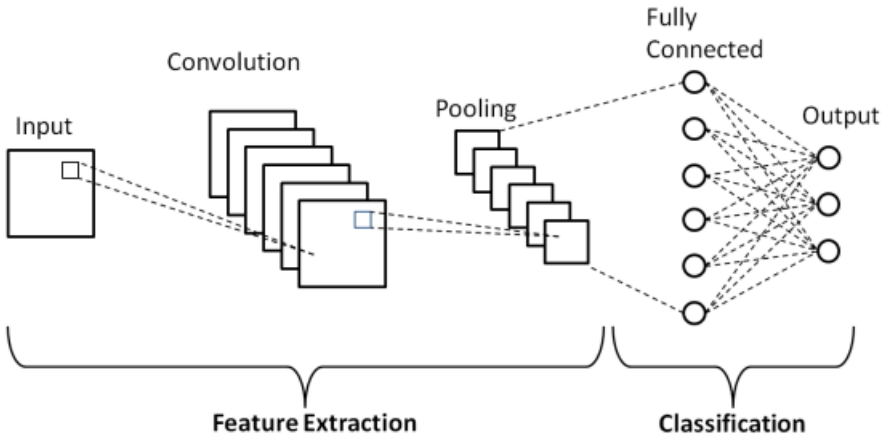
Jakub Bednarz
Krzysztof Madajczak
Mateusz Strzelecki

1. Opis problemu

Przedmiotem projektu była implementacja i analiza konwolucyjnej sieci neuronowej klasyfikującej ludzkie emocje na podstawie obrazów, a także porównanie jej skuteczności z alternatywnymi podejściami, takimi jak inne architektury sieci neuronowych oraz rozwiązania typu transfer learning. Za punkt wyjścia przyjęto uczenie nadzorowane na podstawie wybranego zbioru treningowego FER2013 [1], przedstawiającego zbliżenia na twarze. Zadaniem modelu była klasyfikacja przedstawianych emocji do 7 klas: złości, obrzydzenia, strachu, szczęścia, smutku, zaskoczenia, neutralności. Na tej podstawie stworzony został graficzny interfejs użytkownika umożliwiający wykrywanie twarzy na załadowanych zdjęciach oraz klasyfikację emocji każdej osoby na zdjęciu.

2. Teoretyczny opis użytych metod

Konwolucyjne sieci neuronowe (CNN) są rodzajem sztucznych sieci neuronowych uczenia głębokiego, które zostały zaprojektowane do przetwarzania danych przestrzennych, takich jak obrazy. Podstawą działania CNN są specjalne operacje matematyczne zwane konwolucjami, efektywnie ekstrahujące cechy z obrazów. Wykorzystują one małe macierze (tzw. filtry) do przesuwania się po danych wejściowych. Operacja konwolucji polega na mnożeniu elementów macierzy filtra z odpowiednimi elementami macierzy danych wejściowych, a następnie sumowaniu wyników. Proces ten jest powtarzany dla wszystkich lokalnych obszarów danych wejściowych, tworząc tzw. mapę cech.
Na podstawie odpowiednio złożonych map cech (uzyskanych po wielu warstwach) możliwa jest klasyfikacja obrazów, realizowana przez gęste warstwy w pełni połączone sieci neuronowej.



Rys. 1: Schematyczna architekura klasyfikacji CNN [2]

Kluczowym elementem w konstrukcji CNN jest hierarchiczność następujących po sobie

warstw konwolucyjnych, z których każda ma swoje zadanie. Pierwsze warstwy wykorzystują filtry do wykrywania podstawowych cech, takich jak krawędzie, tekstury i podstawowe wzorce. Kolejne warstwy koncentrują się na coraz bardziej skomplikowanych i abstrakcyjnych cechach, korzystając z wyników poprzednich warstw. Ważne dla efektywnego działania CNN są warstwy poolingowe, które redukują rozmiar map cech, wybierając najważniejsze informacje (najskuteczniejsze są w tym informacje o maksimum cechy).
Konwolucyjne sieci neuronowe są skuteczniejsze w pracy z obrazami niż inne architektury sieci neuronowych ze względu na kilka czynników. Po pierwsze, wykorzystują one lokalne zależności przestrzenne, a więc różne cechy mogą być poprawnie wykrywane niezależnie od siebie. Dzięki temu sieci konwolucyjne są odporne na translacje, co znacznie ułatwia uczenie. Ponadto, wykorzystywanie tego samego filtra na różnych obszarach obrazu znacznie zmniejsza liczbę parametrów do nauki, co sprawia, że sieci są bardziej efektywne obliczeniowo. Występujące po warstwach konwolucyjnych warstwy poolingowe zmniejszają rozmiar danych, zachowując jednocześnie najważniejsze informacje, co pomaga w redukcji overfittingu i poprawie ogólnej wydajności sieci.
Takie podejście pozwala na automatyczne budowanie hierarchii często trudnych do przewidzenia cech obrazu. Wykorzystując wiele warstw, sieci konwolucyjne są w stanie nauczyć się coraz bardziej abstrakcyjnych reprezentacji obrazu, co prowadzi do lepszych wyników w zadaniach takich jak detekcja czy klasyfikacja obrazów.

Jedną z możliwych metod doboru hiperparametrów modelu jest metoda symulowanego wyżarzania. Polega ona na losowych perturbacjach przestrzeni hiperparametrów i akceptowaniu tych które poprawiają działanie modelu. Możliwe jest także akceptowanie parametrów które nie poprawiają funkcji kosztu, w celu uniknięcia utknięcia w lokalnym optimum i szukania lepszych rozwiązań globalnych. Prawdopodobieństwa takiej akceptacji (szerokiego przeszukiwania przestrzeni hiperparametrów) jest określane przez malejącą z czasem temperaturę, na wzór wyżarzania w metalurgii. W dalszych iteracjach bardziej prawdopodobne jest zatem stopniowe doprecyzowanie wartości w ograniczonych przedziałach. Metoda ta nie daje gwarancji znalezienia optymalnego rozwiązania, ale w przeciwieństwie np. do metody Grid Search (sprawdzającej wszystkie kolejne kombinacje) nie wymaga tak ogromnych zasobów obliczeniowych.

Bardzo efektywną metodą uzyskania skutecznego modelu jest tzw. transfer learning, polegający na wykorzystaniu dostępnych bardzo zaawansowanych modeli, trenowanych do rozwiązywania zbliżonego problemu. Istniejące modele, przeważnie bardzo długo uczone na ogromnych zbiorach danych, są adaptowane do nowego zadania poprzez dostosowanie ich wag podczas dodatkowego treningu na docelowym zbiorze. Transfer learning pozwala osiągać świetne wyniki nawet w przypadku nielicznych danych treningowych oraz przyspiesza proces uczenia dzięki wykorzystaniu istniejącej wiedzy.

3. Opis realizacji zadania

Podstawowych narzędziem użytym w projekcie była biblioteka pytorch, pozwalająca na łatwą implementację sieci neuronowych. W celu uzyskania adekwatnych możliwości obliczeniowych przydatne były narzędzia cuda oraz platforma kaggle, umożliwiające wykorzystanie kart graficznych oraz obliczeń w chmurze do przyspieszenia obliczeń. Wykorzystywane zostały również pakiety numpy, torchvision, sklearn, opencv oraz matplotlib.

3.1. Implementacja architektury CNN

Pierwszym etapem było stworzenie konwolucyjnej sieci neuronowej dla analizowanego problemu klasyfikacji wieloklasowej. Przyjęta została następująca architektura:

- łącznie 6 warstw konwolucyjnych, hierarchicznie wykrywających coraz większe struktury na zdjęciu
- po każdej warstwie konwolucyjnej następuje warstwa aktywacji ReLu, która wprowadza nieliniowość do modelu
- warstwy normalizacji wsadowej stabilizujące model po warstwie aktywacji poprzez redukcje wewnętrznych zmian kowariancji
- co dwa bloki warstw konwolucyjnych stosowana jest warstwa max pooling okna 2x2, która zmniejsza wymiary przestrzenne danych wejściowych, wybierając maksymalną wartość w obrębie okna poolingowego
- warstwy dropout, losowo ustawiające część jednostek wejściowych na zero podczas treningu, co pomaga zapobiegać przeuczeniu przez redukcję współzależności pomiędzy neuronami
- warstwy w pełni połączone zastosowane w celu klasyfikacji emocji na podstawie wykrytych cech

Danymi dla sieci były poddane podstawowej obróbce zdjęcia z podstawowego zbioru FER2013 (około 28000 zdjęć treningowych i 7000 testowych), z wydzielonymi 20% danych treningowych na dane walidacyjne wykorzystywane do oceny modelu podczas nauki.

3.2. Poprawa skuteczności modelu: augmentacja danych, dobór hiperparametrów

Dla poprawy skuteczności modelu kluczowe było przede wszystkim rozszerzenie zbioru treningowego poprzez augmentację danych. Pozwala ona na generalizację modelu, przez co może on sobie lepiej radzić z nieznanymi danymi testowymi. W tym celu zbiór został rozszerzony o odbicia lustrzane oraz zdjęcia poddane rotacji. Zbiór testowy pozostał niezmieniony. W ten sposób liczba dostępnych danych treningowych została zwiększona z około 28 tysięcy do ponad 114 tysięcy obrazów.

W celu poprawy doboru stałej uczenia, liczby epok oraz rozmiaru wsadu zastosowana została metoda symulowanego wyżarzania. Dla pierwszych dwóch parametrów określone zostały przedziały poszukiwań oraz wielkość możliwego kroku. Rozmiar wsadu był losowany z zbioru możliwych wartości. Ze względu na możliwości obliczeniowe, liczba iteracji została określona na 30.

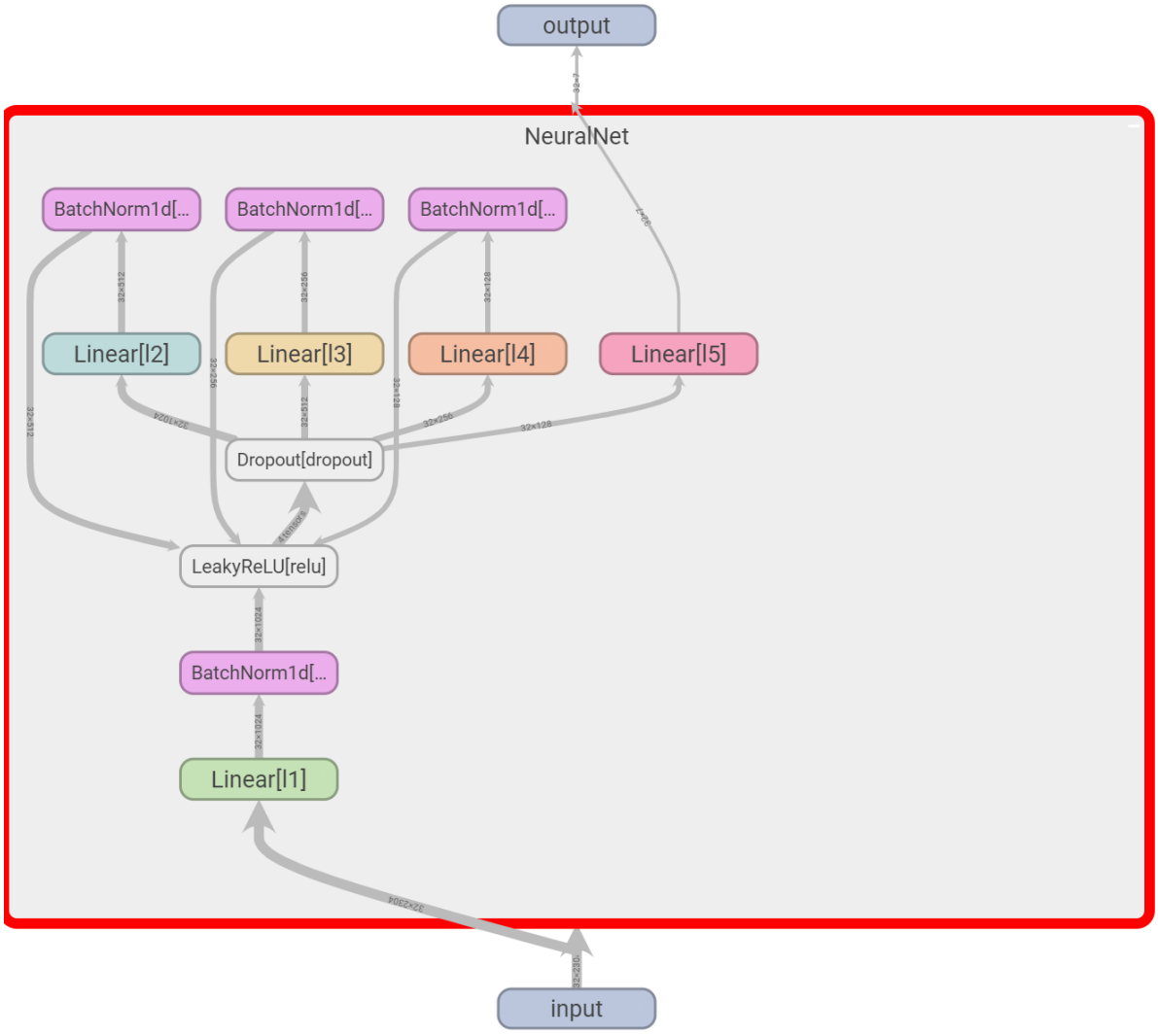
3.3. Porównawczy model feedforward

Dla porównania wybraliśmy model feedforward metoda uczenia maszynowego która korzysta która umożliwia propagacje sygnałów w jednym kierunku: od warstwy wejściowej przez warstwę ukrytą do warstwy wyjściowej.

1. Definicja warstw: W metodzie feedforward tworzymy kolejne warstwy sieci neuronowej. Każda warstwa składa się z neuronów, które wykonują operacje na danych wejściowych i przekazują wyniki do kolejnej warstwy. W metodzie FF korzystamy z warstwy gęstej zwanej też jako (fully connected layer).
2. Funkcje aktywacji: W metodzie FF zastosowaliśmy również metodę ReLU jednakże pokusiliśmy się o eksperymenty i przetestowaliśmy również metodę Leaky ReLU i poprawiła ona wyniki.
3. Ważenie i sumowanie: W metodzie feedforward każdy neuron otrzymuje dane wejściowe z poprzedniej warstwy, waży te dane z odpowiednimi wagami i sumuje je. Proces ten polega na pomnożeniu danych wejściowych przez wagi neuronu i

zsumowaniu wyników.

4. Inicjalizacja wag: Inicjalizacja wag w sieciach neuronowych jest ważnym etapem, który może mieć wpływ na skuteczność uczenia się sieci. Istnieje wiele metod inicjalizacji wag, takich jak inicjalizacja losowa, inicjalizacja Xavier itp. My przetestowaliśmy 2 wymienione i metoda Xavier umożliwiła poprawę wyników w tej metodzie o 0.2 %. W celu poprawy skuteczności zastosowaliśmy również dropouty które zerują niektóre neurony i dzięki temu zapobiegają overfitingowi.
5. Propagacja w przód (Forward propagation): Główna idea metody feedforward polega na przekazywaniu danych przez sieć od warstwy wejściowej, przez warstwy ukryte, aż do warstwy wyjściowej. Przy forward propagation każda warstwa przetwarza dane z poprzedniej warstwy i przekazuje je do kolejnej warstwy, aż do uzyskania wyniku na warstwie wyjściowej. W naszym przypadku zdjęcie miało początkowy rozmiar 2304 co umożliwiło zastosować 5 warstw ukrytych o wielkości odpowiednio 1024, 512, 256, 128,7



Model sieci Feedforward wygenerowanej za pomocą tensorboardX

3.4. Zastosowanie tranfer learningu

W przypadku transfer learningu (TL) skorzystaliśmy z wyuczonych modeli dostępnych w bibliotece torchvision, takich jak znana AlexNet, VGG, ResNet, itd. W naszej pracy skoncentrowaliśmy się na rodzinie modeli ResNet. Rodzina modeli ResNet (Residual Network) wprowadza koncepcję resztkowych połączeń. Składa się z różnych wariantów, takich jak ResNet18, ResNet34, ResNet50, itd., które charakteryzują się głębokimi architekturami z blokami resztkowymi.

Najwięcej uwagi poświęciliśmy modelowi ResNet18, który został dostosowany do pracy z danymi w trzech kanałach RGB. Jednak nasze dane treningowe były w formacie przestrzeni szarości, dlatego dane walidacyjne, które zawierały kilka kanałów kolorystycznych, mogłyby osiągnąć lepsze rezultaty w przypadku uczenia na zestawie danych kolorowych.

ResNet-18 składa się z 18 warstw, w tym 17 warstw konwolucyjnych i jednej warstwy w pełni połączonej (fully connected). Architektura modelu składa się z następujących elementów:

Bloki resztkowe: ResNet-18 wykorzystuje cztery bloki resztkowe, które składają się z warstw konwolucyjnych oraz połączeń skrótowych, umożliwiających bezpośrednie przekazywanie informacji między warstwami. To pomaga w efektywnym uczeniu się sieci neuronowej.
Niestety, model ResNet34 okazał się zbyt wymagający obliczeniowo dla dostępnych przez nas zasobów sprzętowych.

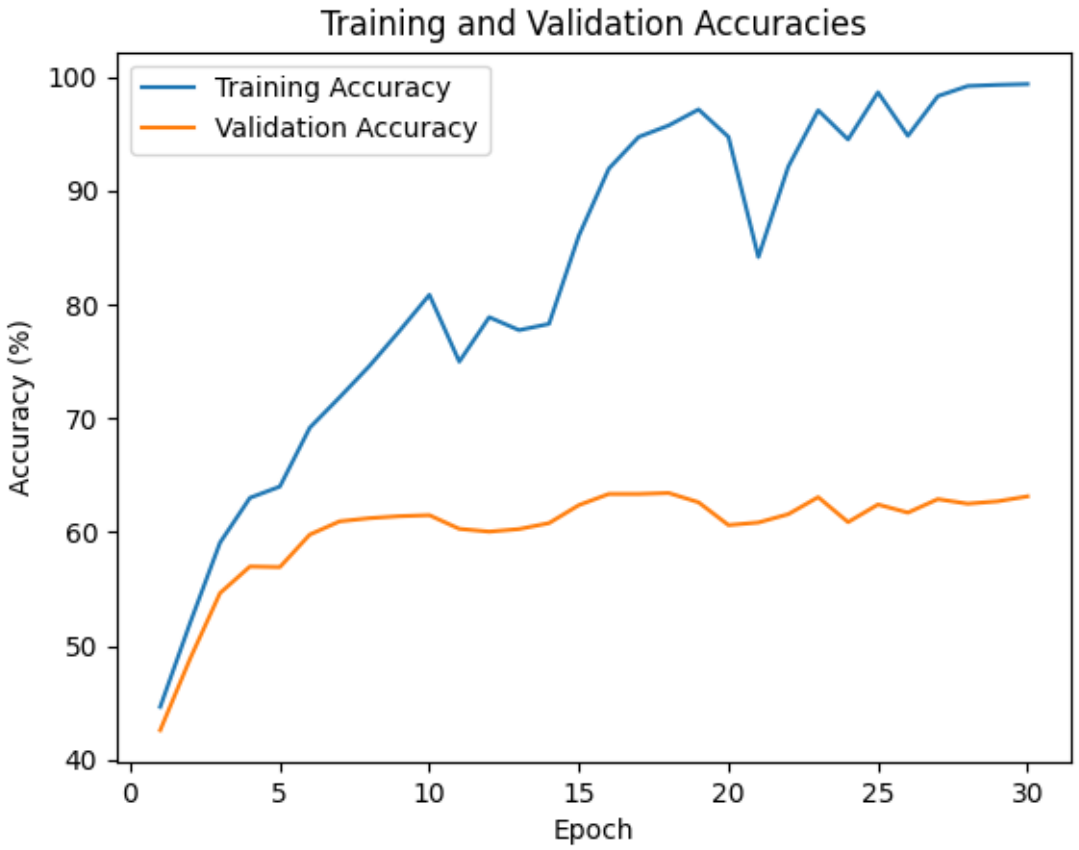
3.5. Stworzenie interfejsu użytkownika

Interfejs został stworzony przy użyciu biblioteki tkinter, która umożliwia użytkownikowi wybór obrazu, detekcję twarzy i rozpoznawanie emocji. W implementacji interfejsu wykorzystano również bibliotekę OpenCV, która używana jest do operacji na obrazach, takich jak konwersja kolorów, zmiana rozmiarów, rysowanie na obrazie oraz detekcja twarzy i oczu. Kaskada Haara pozwala na wykrywanie twarzy i oczu. Algorytm sprawdza, czy w wykrytych twarzach znajdują się oczy. Twarze te są zaznaczane na obrazie, a następnie przycinane i przechowywane do analizy emocji. W analizie emocji wykorzystano wcześniej zaimplementowane modele: CNN, Feed Forward oraz Transfer Learning. Wyniki analizy emocji są zapisywane na obrazie.

4. Prezentacja osiągniętych wyników

4.1. Implementacja architektury CNN

Podczas treningu sieci w każdej epoce mierzona była skuteczność aktualnego modelu na zbiorze treningowym i walidacyjnym. Przebieg tych wartości przedstawia Wyk.1:

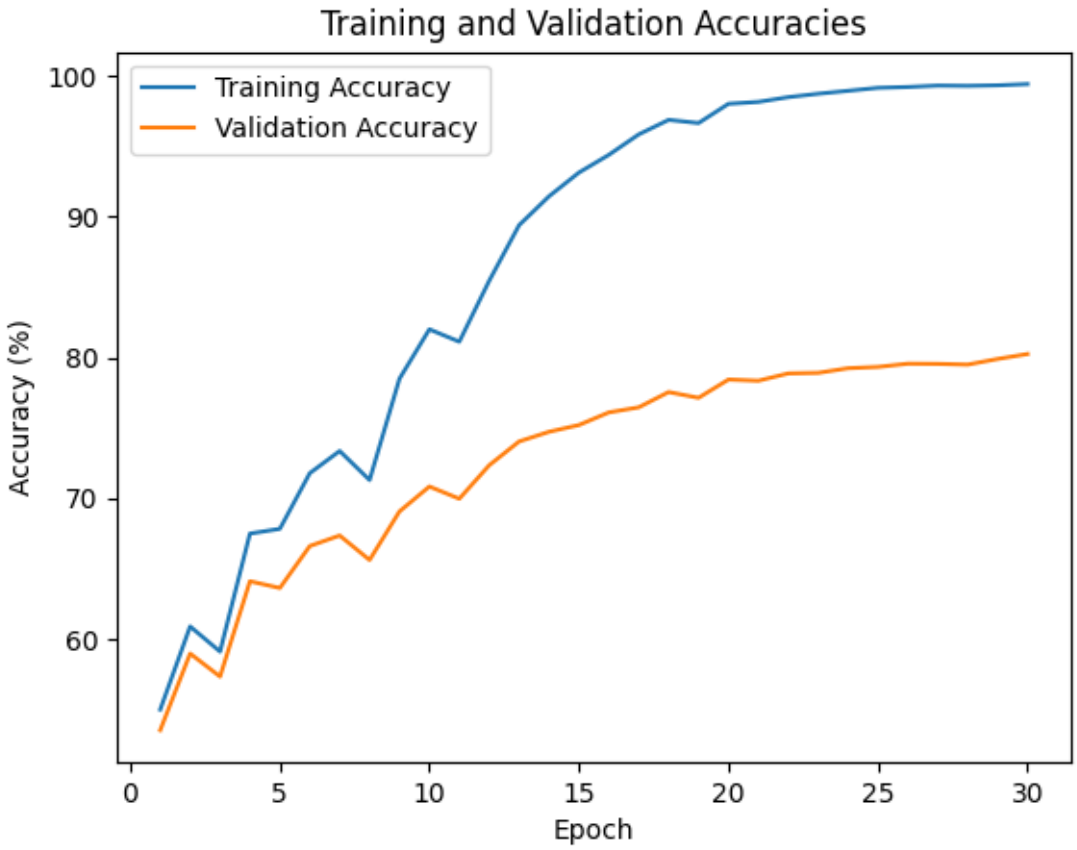


Wyk. 1: Wykres dokładności treningowej i walidacyjnej podstawowego modelu

Uzyskany model uzyskał skuteczność testową na poziomie 63,78%.
Obliczone zostały także podstawowe metryki przewidywań modelu dla każdej klasy: macierz konfuzji, precyzja, odzysk oraz wynik F1, jak również ich średnie ważone. Należy odnotować, że wszystkie średnie ważone okazały się być bardzo zbliżone [6].

4.2. Poprawa skuteczności modelu: augmentacja danych, dobór hiperparametrów

Analogiczne działania dla poprawionego modelu pokazują postęp modelu:

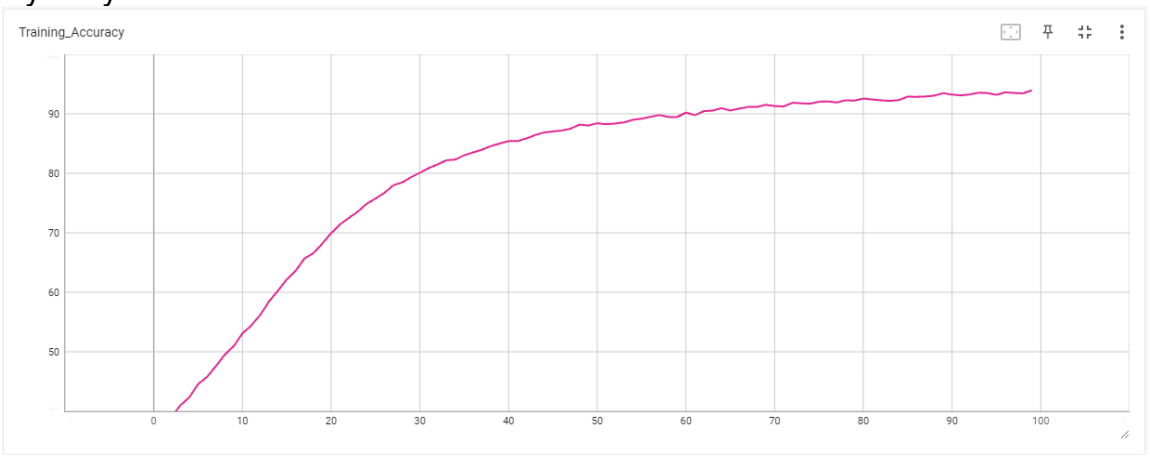


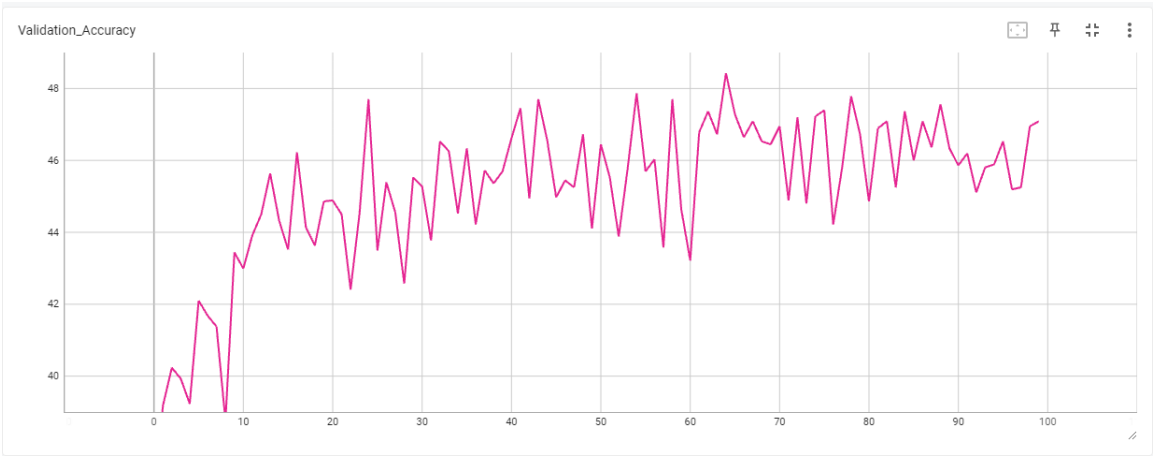
Wyk. 2: Wykres dokładności treningowej i walidacyjnej modelu ulepszonego

Poprawiony model uzyskał skuteczność testową na poziomie 68,61%.
Po wykonaniu dodatkowych porównawczych, liczba epok trenowania modelu została podniesiona względem wartości wskazywanej przez wyżarzanie.
Wszystkie uwzględnione metryki wykazały podobne zachowanie jak dla modelu podstawowego, z odpowiednio wyższymi wartościami [10].

4.3. Porównawczy model feedforward

Podczas treningu sieci w każdej epoce mierzona była skuteczność aktualnego modelu na zbiorze treningowym i walidacyjnym. Przebieg tych wartości przedstawiają poniższe wykresy:

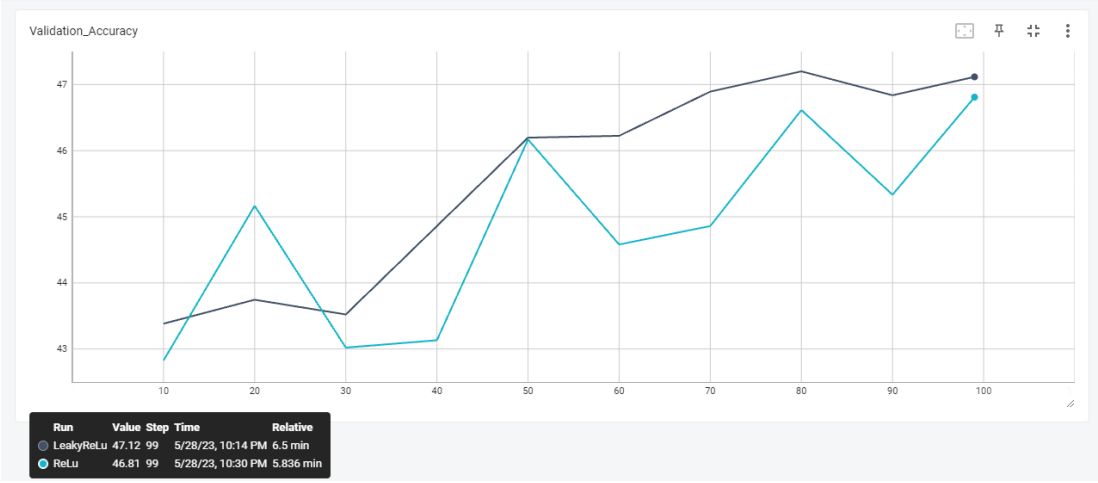




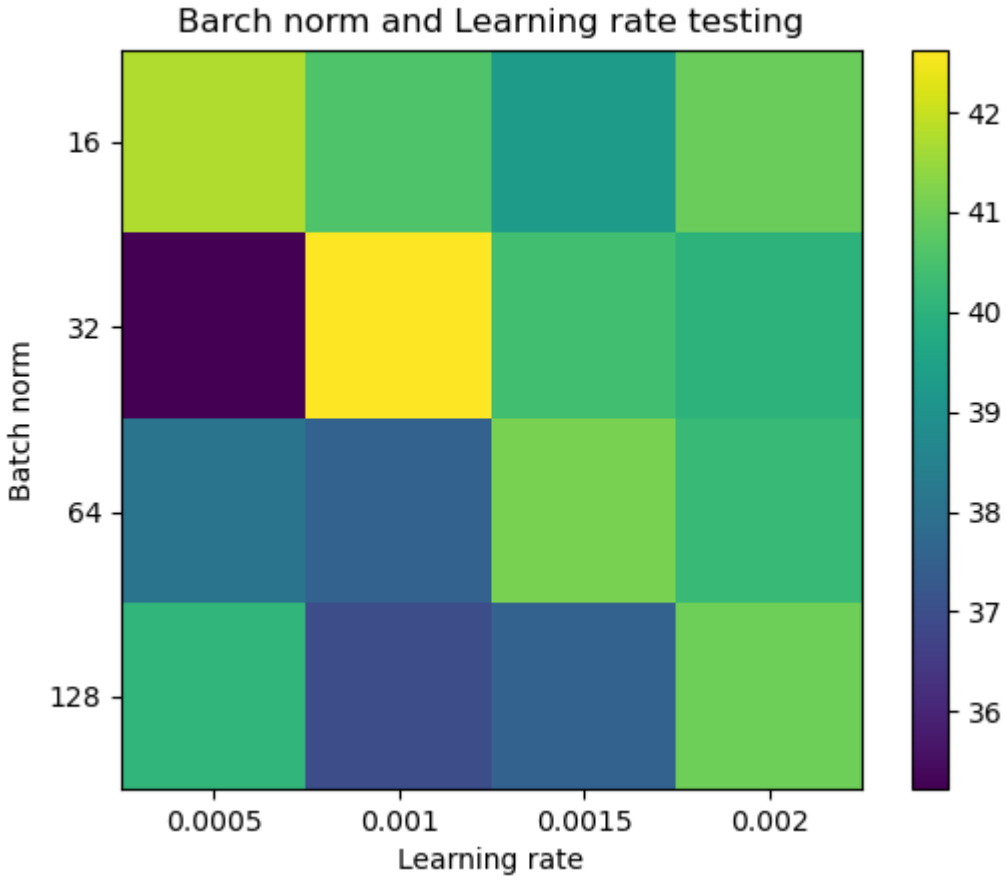
Model uzyskał skuteczność 47.08 % na 100 epoce jednakże można zauważyć niż przy 65 epoce uzyskał lepszy 48.42% wynik.

W celu sprawdzenia czy możliwe jest otrzymanie lepszych wyników zostały przeprowadzone testy gdzie zmienialiśmy hiper-parametry jak i też inne elementy wpływającą na skuteczność modelu.

- Leaky ReLu vs ReLu

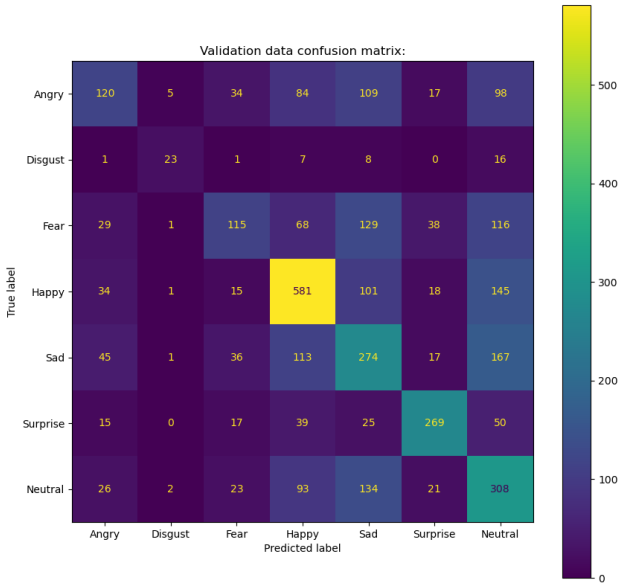


- Różne wartości Learning Rate jak i batch norm

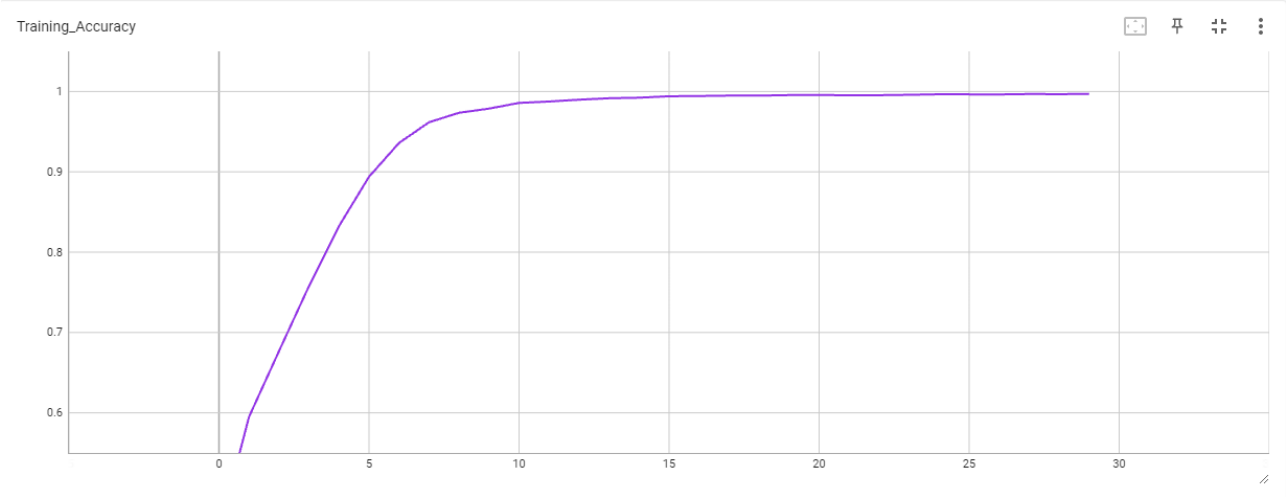


Testy zostały w tym przypadku przeprowadzone na 5 epokach w celach poglądowych Jak można zauważyć pierworodnych wybór hiper-parametrów okazał się prawidłowy

W celu oceny jakości normalizacji danych, przygotowaliśmy macierz konfuzji walidacyjnej, aby zobrazować występowanie błędów związanych z poszczególnymi cechami. Warto zauważyć, że w naszym przypadku występuje znaczna nadreprezentacja danych dotyczących cechy "happy", co może wpłynąć na naukę modelu, skłaniając go do częstszego rozpoznawania tej cechy. Jednocześnie obserwujemy podreprezentację cechy "Disgust", co znacząco zmniejsza szanse jej poprawnego rozpoznania.

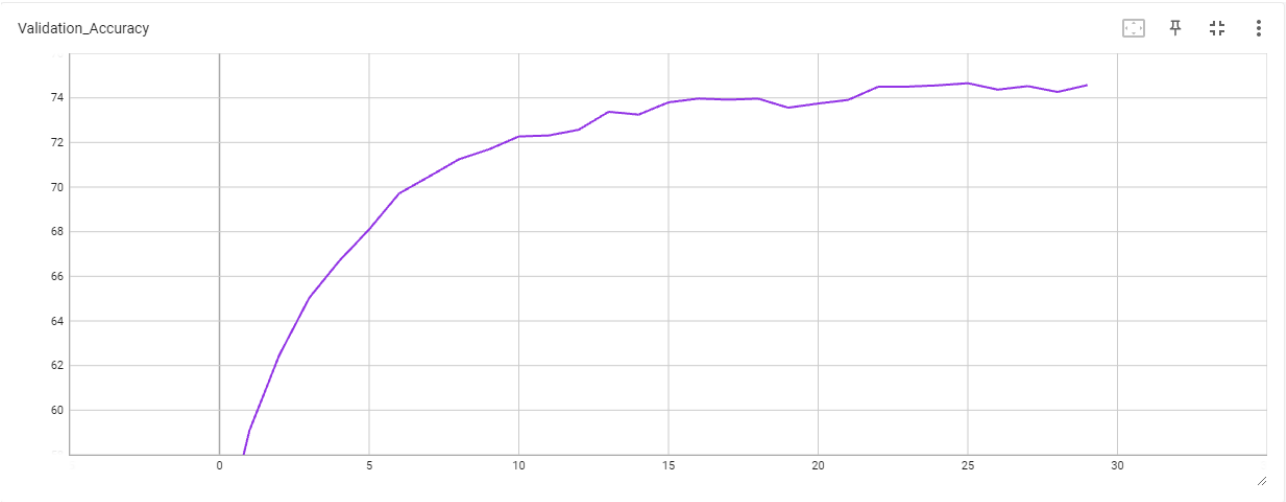


4.4. Zastosowanie tranfer learningu



Jak można zauważyć nasz model po niewielu epokach osiągnął bardzo dużą skuteczność na danych treningowych. Na ostatniej epoce model osiągnął skuteczność 99.97%.

batch_size	Learning rate	SGD momentum
128	0.001	0.9



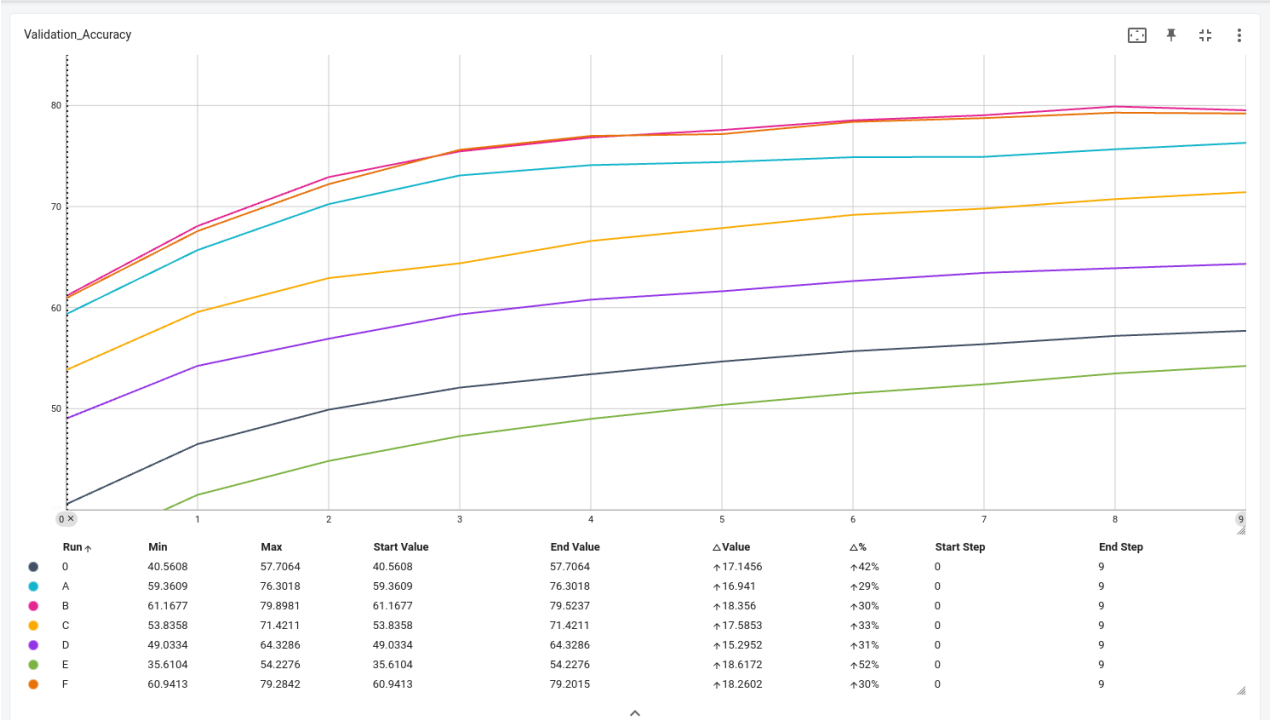
Maksymalna skuteczność osiągnięta na danych walidacyjnych wyniosła 74,66%, co jest wynikiem wysokim w porównaniu do innych metod. Jednak postanowiliśmy przeprowadzić dodatkowe testy w celu sprawdzenia, czy dobór odpowiednich hiperparametrów może poprawić ten wynik. Wykorzystaliśmy metodę random search.

Na danych testowych uzyskaliśmy skuteczność wynoszącą 61,52%.

Podczas eksperymentów zastosowaliśmy różne rodzaje optymalizatorów, takie jak Stochastic Gradient Descent (SGD) z wartością momentum ustawioną na 0,9 oraz optymalizator Adam z domyślnymi wartościami parametrów beta (0,9, 0,999). Przetestowaliśmy również różne wartości współczynnika uczenia (learning rate) oraz rozmiary paczek (batch size), tj. 126 i 256.

ID	Optimalizator	Learning rate	Batch size	Skuteczności walidacji
0	adam	0.00001	256	58.21%
A	adam	0.0001	256	76.30%
B	adam	0.0001	126	79.52%

C	sgd	0.001	126	71.42%
D	sgd	0.001	256	64.32%
E	sgd	0.0001	128	54.27%
F	Adam W	0.0001	128	79,20%



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
0	40.5608	57.7064	40.5608	57.7064	↑17.1456	↑42%	0	9
A	59.3609	76.3018	59.3609	76.3018	↑16.941	↑29%	0	9
B	61.1677	79.8981	61.1677	79.5237	↑18.356	↑30%	0	9
C	53.8358	71.4211	53.8358	71.4211	↑17.5853	↑33%	0	9
D	49.0334	64.3286	49.0334	64.3286	↑15.2952	↑31%	0	9
E	35.6104	54.2276	35.6104	54.2276	↑18.6172	↑52%	0	9
F	60.9413	79.2842	60.9413	79.2015	↑18.2602	↑30%	0	9

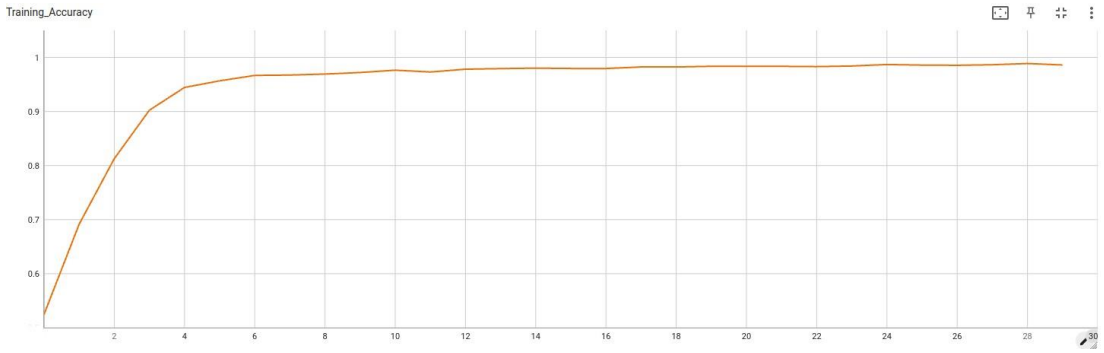
Emocja	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Dokładność [%]	69.66%	65.29%	62.63%	86.73%	68.42%	84.36%	71.57%

Po wykonaniu random search'a model o najlepszych parametrach (B) został wytrenowany na dłuższą liczbę epok (30) by porównać jego skuteczność z pozostałymi modelami. Parametry treningu lr = 0.001 batch_size = 126 oraz optymalizator adam.

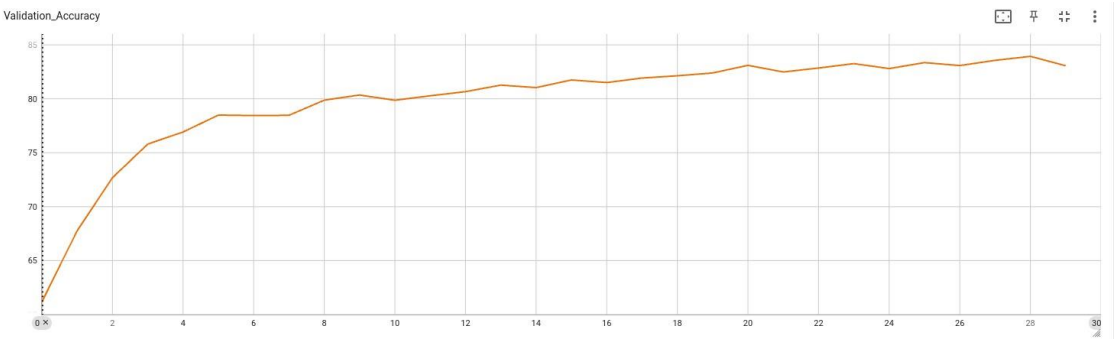
Oto wyniki:

	Dokładność Walidacyjna	Dokładność Testowa
Dokładność [%]	83.06	63.36

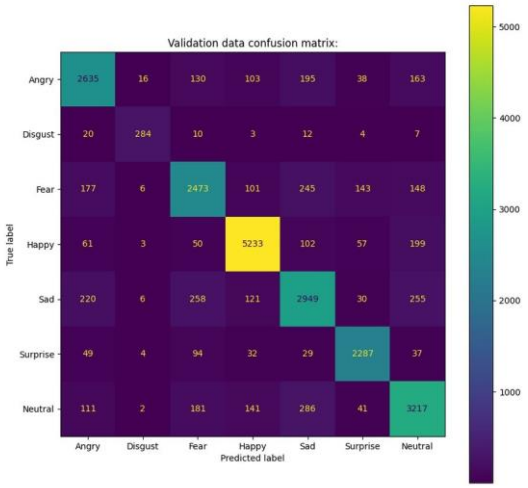
Dokładność Treningowa po 30 epokach (acc / epoch):

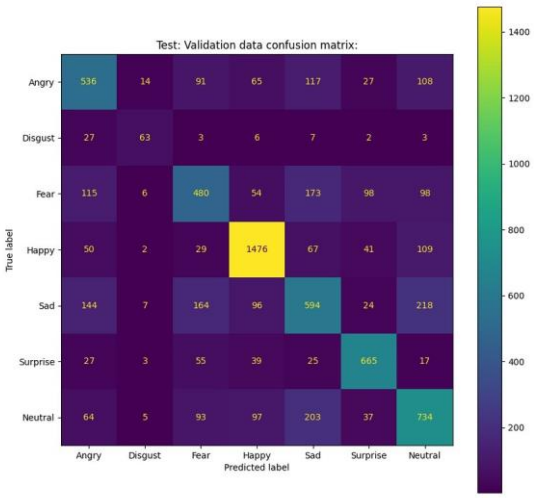


Dokładność Walidacyjna po 30 epokach (acc / epoch):

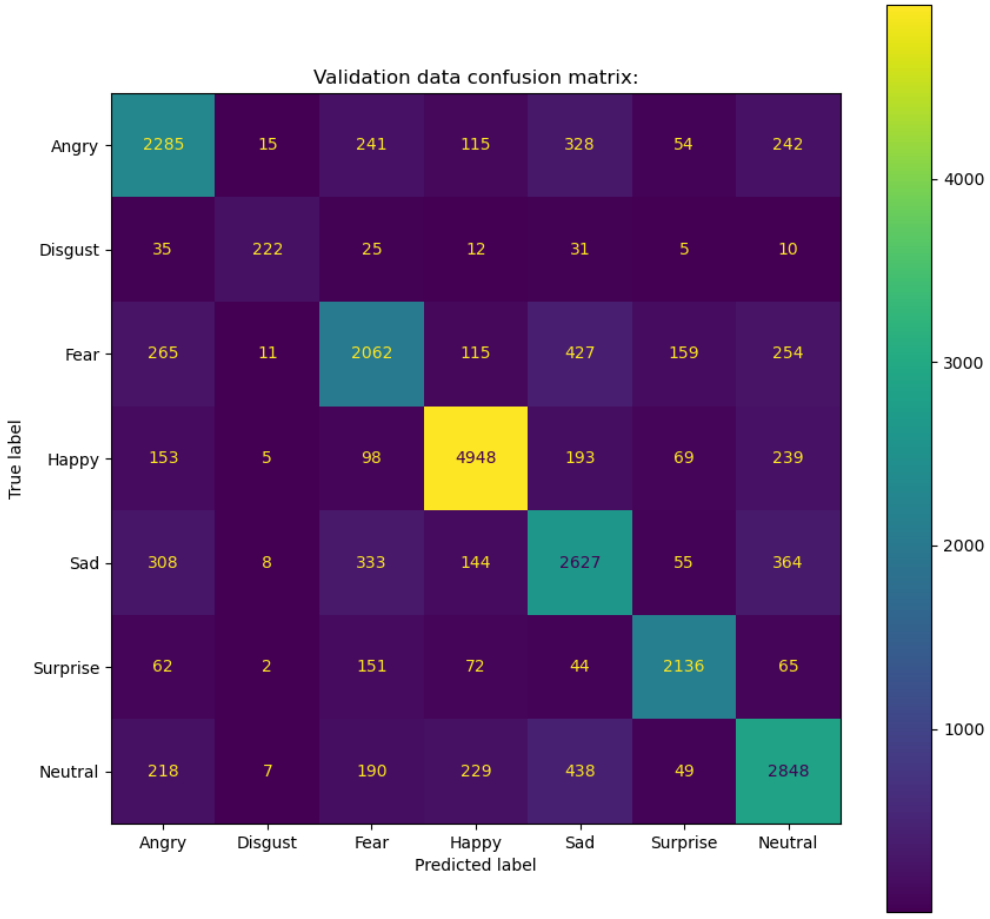


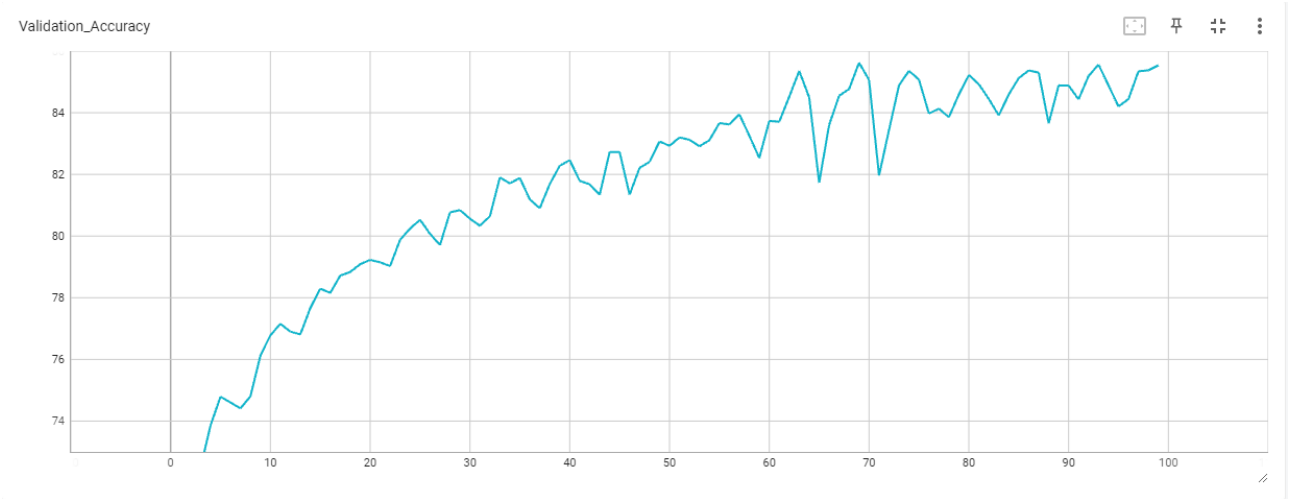
Macierze pomyłek:



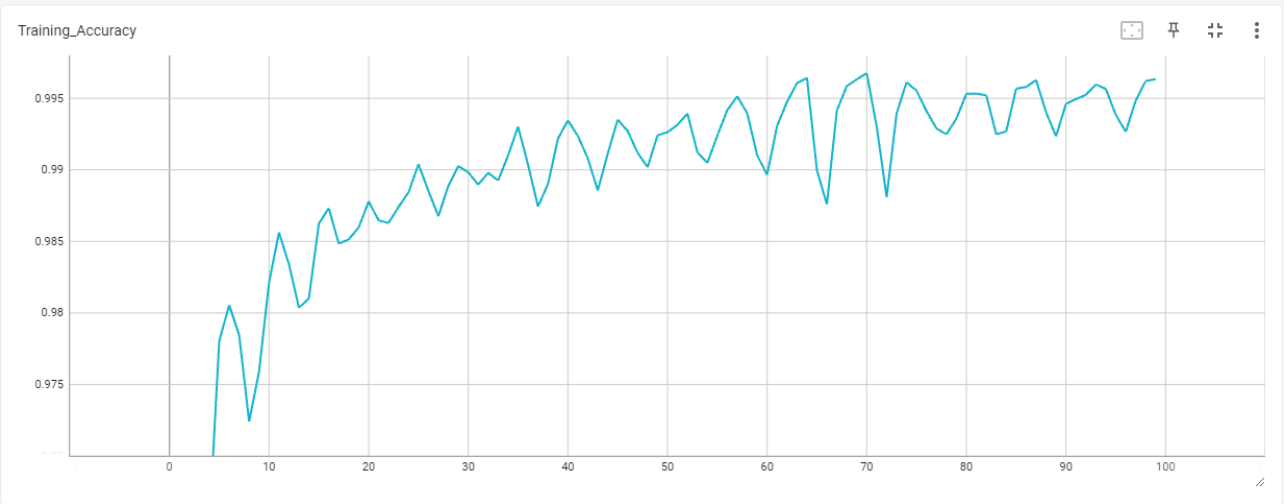


W podowodu długie czasu trenowania pokusiliśmy się tylko o 1 trening na 100 epokach.





Skuteczność na danych walidacyjnych osiągnęła pułap 85.6%



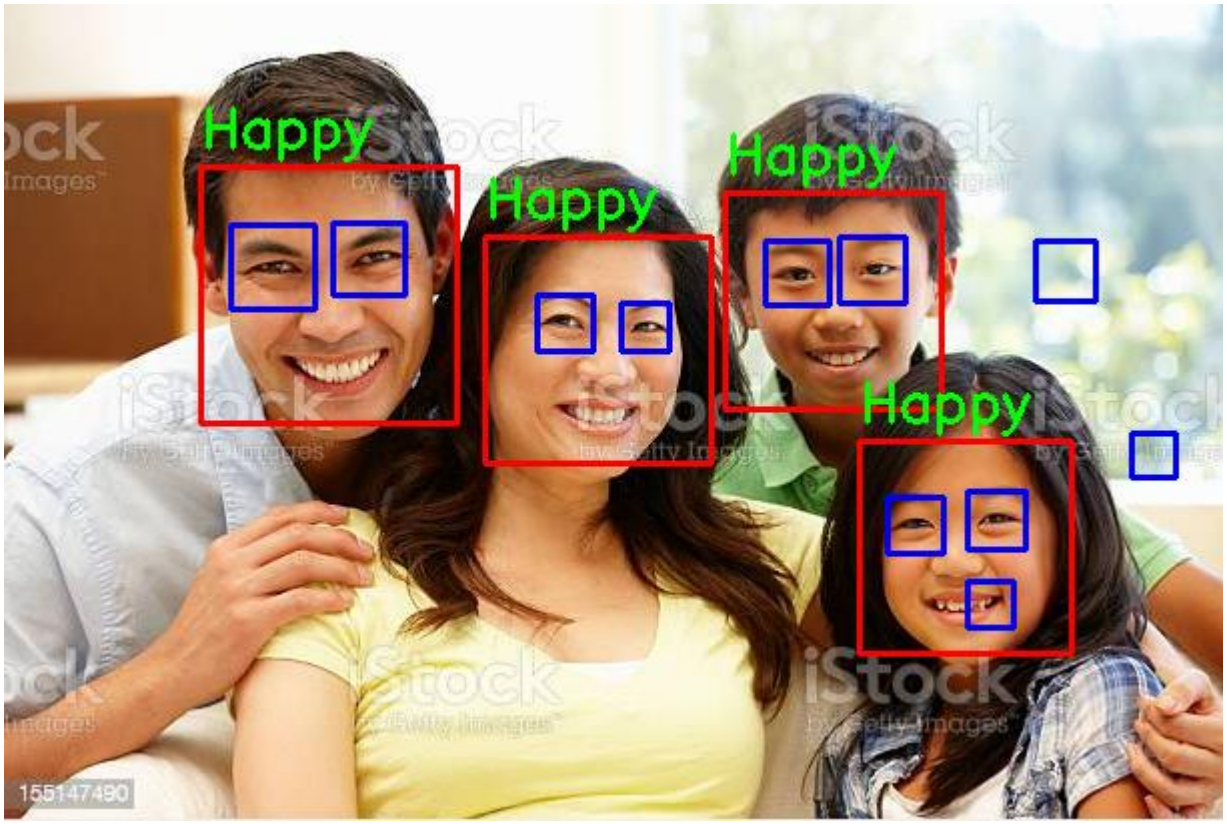
Skuteczność treningowa podobnie jak w przypadku wcześniejszym bardzo szybko osiągnęła wysoki próg.

Na danych testowych osiągnął skuteczność 64.36%.
Porównując skuteczność walidacyjną do skuteczności testowej można zadać pytanie skąd wynika aż taka różnica w dokładności.
Ta różnica w dokładności między wynikami treningowymi a testowymi może wynikać z overfittingu. Overfitting jest sytuacją, gdy model nauczył się zbyt dokładnie odwzorować dane treningowe.

Co pokazuje że nasze rozwiązanie za pomocą sieci CNN jest lepsze.

4.5. Stworzenie interfejsu użytkownika

Poniżej znajduję się przykładowe zdjęcie, na którym przedstawione są wykryte emocje na twarzach:



5. Dyskusja

Uzyskiwane przez nasze trzy modele skuteczności są zgodne z tym czego należało się spodziewać na podstawie obecnego stanu wiedzy i osiągnięć [5] odnośnie przedstawionego problemu.
Podstawowy wynik konwolucyjnej sieci neuronowej na poziomie nieznacznie przekraczającym 63% jest znacznie lepszy niż wynik uzyskanego przykładową inną siecią neuronową, która była tutaj sieć typu feedforward. Dobrze ilustruje to przewagi sieci konwolucyjnych nad innymi architekturami, opisane w części teoretycznej. Głównym czynnikiem, który był w stanie wydatnie poprawić skuteczność sieci CNN była augmentacja danych treningowych. Pozwoliła ona na lepszą generalizację obserwacji cech, co poprawiło wyniki na nieznanych danych. Postęp modelu dobrze widać po o wiele dłuższym utrzymywaniu rosnącej tendencji przez krzywą skuteczności na zbiorze walidacyjnym niż miało to miejsce dla podstawowego modelu.
Największą trudnością w dalszej poprawie skuteczności sieci były ograniczone możliwości obliczeniowe. Najbardziej widoczne było to w przypadku problemu doboru hiperparametrów. Ze względu na ogromną złożoność obliczeń bezpośredniego przeszukiwania przestrzeni możliwości (Grid Search), zdecydowaliśmy się na implementację metody symulowanego wyżarzania. Również tutaj jednak, ze względu na ograniczenie ilości iiteracji, pula dopasowywanych hiperparametrów była dość mała. W związku z powyższym, wiele czynników (ilość filtrów poszczególnych warstw, rozmiary jądra, dropouty, metoda aktywacji) musiało być dobranych heurystycznie, na podstawie typowo stosowanych wartości [3][7].
Niemniej jednak, osiągnięty wynik na poziomie 68,61% nie jest odległy od najlepszych osiąganych przez CNN na tym zbiorze rezultatów [3][8][9], mieszczących się w przedziale 71-75%.
Warto tutaj odnotować, że jest to dość wymagający zbiór danych, a ludzka zdolność do klasyfikacji emocji z obrazów FER2013 jest szacowana jedynie na 65±5% [8][9].

Lepsze rezultaty są możliwe do osiągnięcia korzystając z metod transfer learningu.

Niestety, w naszym wybranym modelu napotkaliśmy duży problem z przeuczeniem (overfitting). Nie udało nam się uzyskać lepszych wyników, ponieważ model bardzo szybko dostosowywał się do danych treningowych, co skutkowało niską skutecznością predykcji na zbiorze testowym. Chociaż zastosowanie większej liczby dropoutów oraz wprowadzenie ograniczeń wag modelu mogłoby znacznie poprawić dokładność predykcji, oznaczałoby to znaczącą ingerencję w sam model, co jest sprzeczne z ideą transfer learningu.

Warto wspomnieć niż dane na których operowaliśmy miały były niezbalansowane, przez co nasza sieć miała w każdym przypadku trudniejsze zadanie w poprawnym interpretowaniu niektórych emocji.

Kaskada Haara pozwala na wykrywanie twarzy i oczu, ale jest podatna na błędne detekcję. W niektórych przypadkach, kaskada wykrywa twarze i oczy tam gdzie ich nie ma.

Bibliografia:

[1] dane na podstawie: https://www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/data
[2] źródło schematu: https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909
[3] Deep Learning Praca z językiem Python i biblioteką Keras, François Chollet, Helion 2019
[4] https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/
[5] https://www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/overview
[6] obliczone metryki znajdują się w pliku logs/metrics/basic_cnn_metrics.txt
[7] https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
[8] https://arxiv.org/ftp/arxiv/papers/2105/2105.03588.pdf
[9] http://cs230.stanford.edu/projects_winter_2020/reports/32610274.pdf
[10] obliczone metryki znajdują się w pliku logs/metrics/final_cnn_metrics.txt