Warsztaty z Technik Uczenia Maszynowego

Piotr Rowicki,Igor Staręga

7 czerwca 2024

Tabela Zmian

data	zmiana
08.03	dodanie dokumentacji wstępnej
19.03	pełen opis sekcji 2

1 Dokumentacja Wstępna

1.1 Opis problemu

Zagadnieniem którym będziemy zajmować się w naszym projekcie jest znajdywanie punktów charakterystycznych twarzy. Jest to jeden z problemów konkursowych dostępnych na platformie Kaggle. Dokładniej polegać on będzie na znalezieniu pozycji takich części twarzy jak oczy, uszy, nos, usta (wstępnie na zdjęciach, docelowo w czasie rzeczywistym na obrazie z kamery).

1.2 Zbiór danych

Do wytrenowania sieci wykorzystamy zbiory danych udostępnione na wcześniej wspomnianej platformie,

1.3 Użyte narzędzia

Językiem użytym do implementacji będzie Python. Do obróbki danych zostanie wykorzystana biblioteka Pandas, co wynika z natury tego, jak dane zostały nam przekazane. Do implementacji modelu planujemy wykorzystać bibliotekę TensorFlow, a jeżeli pozwoli na to czas, również PyTorch (do implementacji alternatywnego modelu). Do implementacji aplikacji graficznej wykorzystamy również bibliotekę OpenCV, w celu wykrywania twarzy. Decyzję o Frameworku do samego GUI zostawiamy na później

1.4 Podział pracy

1. Piotr Rowicki
   (a) Analiza obróbka danych
   (b) możliwa implementacja drugiego modelu
   (c) wspólna implementacja szaty graficznej
2. Igor Staręga

(a) Implementacja pierwszego modelu

(b) wspólna implementacja szaty graficznej

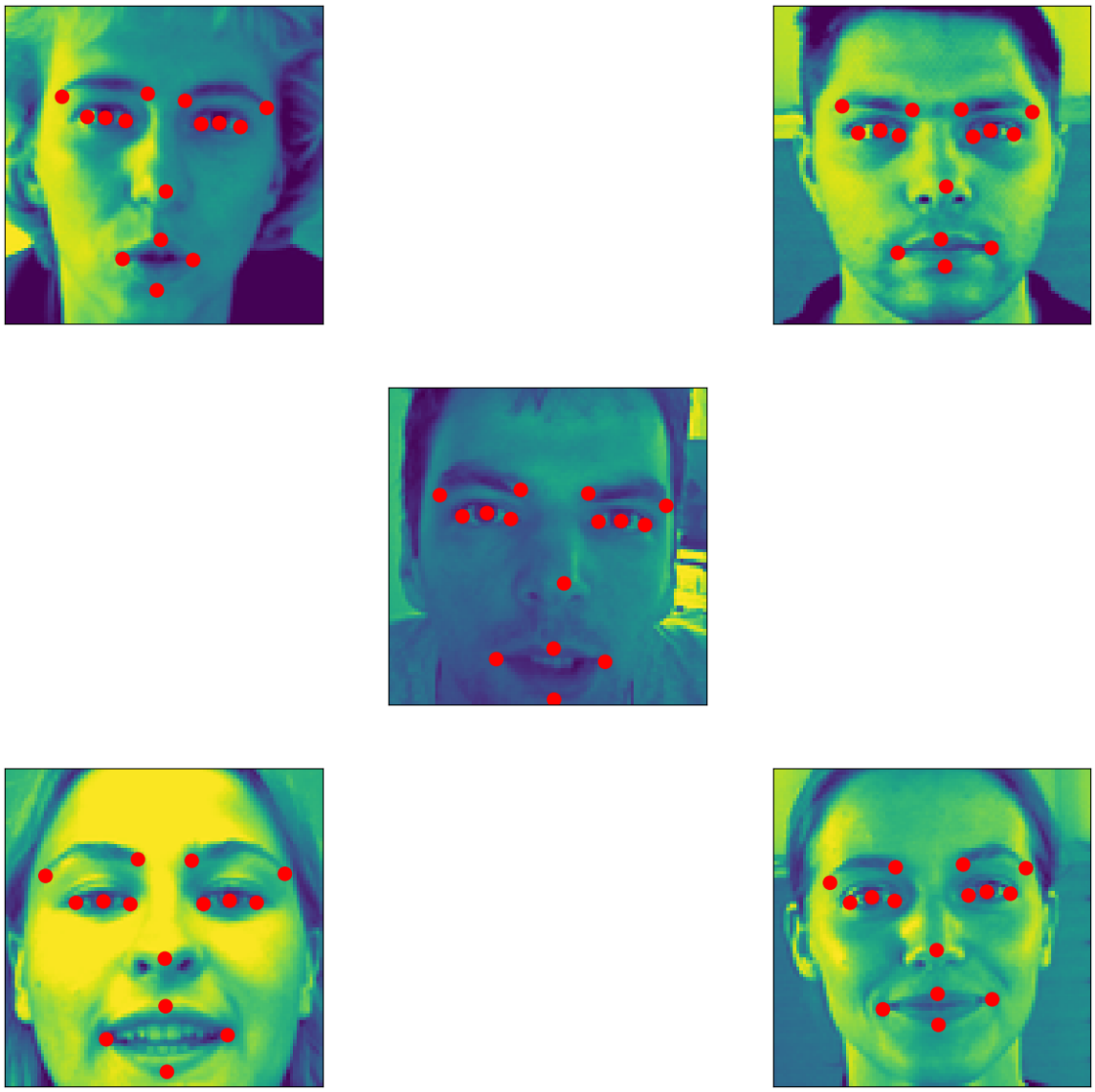
2 Analiza i Obróbka Danych

2.1 Opis zbioru

Dane dostępne są w postaci tabeli zbudowanej z 31 kolumn i 7049 wierszy. Kolumna 31 zawiera ciąg liczb stałoprzecinkowych reprezentujących monochromatyczny obraz twarzy. rozdzielczość tych zdjęć to 96 na 96 pikseli. pozostałe 30 kolumn zawiera współrzędne charakterystycznych cech twarzy(każda cecha ma poświęcone dwie kolumny, na współrzędną x-ową współrzędną i y-ową). Rozważane cechy to:

1. środek lewego oka
2. zewnętrzny kącik lewego oka
3. wewnętrzny kącik lewego oka
4. środek prawego oka
5. zewnętrzny kącik prawego oka
6. wewnętrzny kącik prawego oka
7. czubek nosa
8. zewnętrzny koniec lewej brwi
9. wewnętrzny koniec lewej brwi
10. zewnętrzny koniec prawej brwi
11. wewnętrzny koniec prawej brwi
12. lewy kącik ust
13. prawy kącik ust
14. środek górnej wargi
15. środek dolnej wargi

kilka wybranych zdjęć, wraz z nałożonymi na nie cechami przedstawione zostały na rysunku 1



Rysunek 1: Zdjęcia przedstawiają kilka wybranych obrazów ze zbioru danych wraz z nałożonymi współrzędnymi kluczowych cech

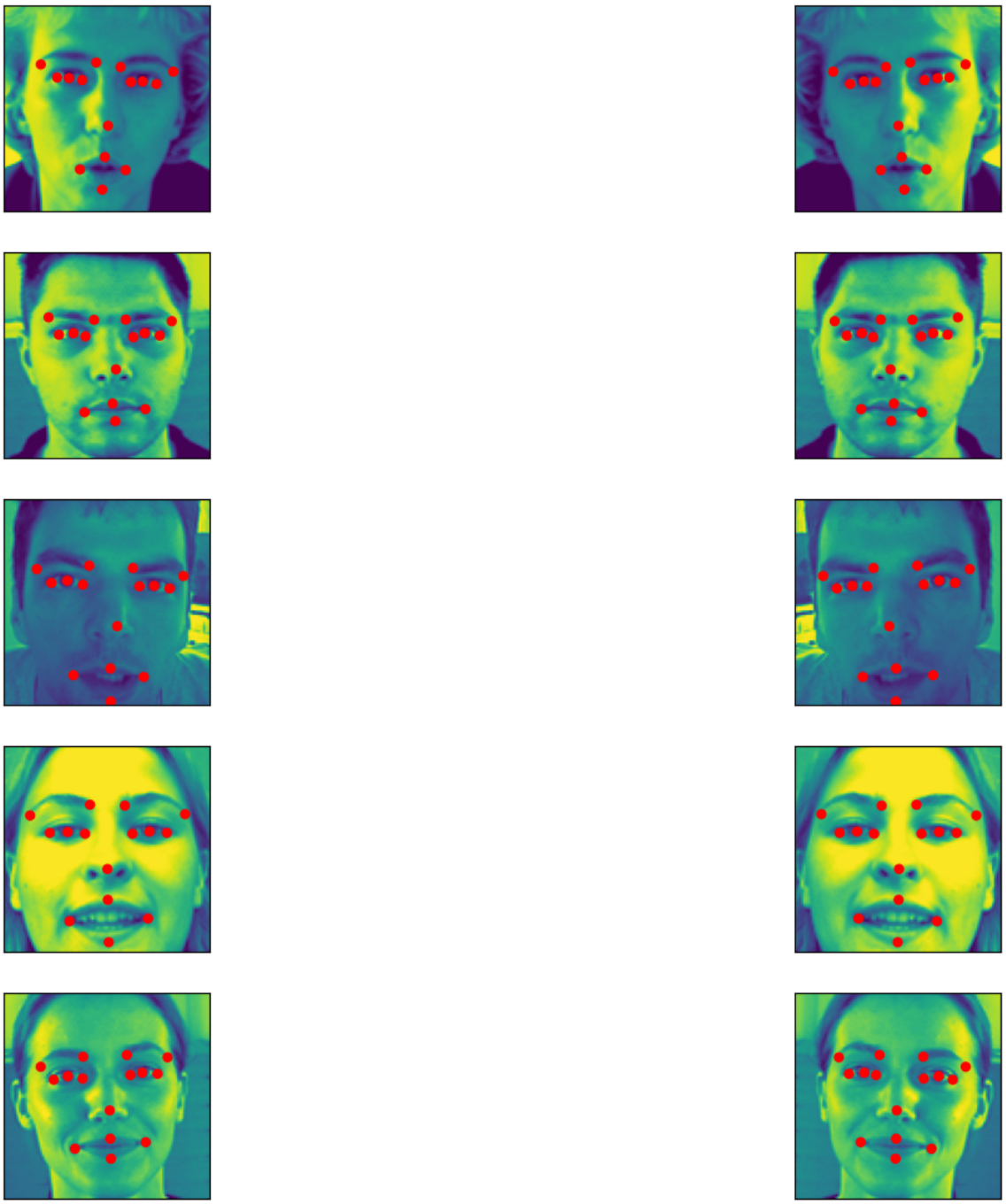
2.2 obróbka danych

2.2.1 problemy ze zbiorem

Niestety w zbiorze danych znajduję się wiele wierszy, które nie mają wszystkich danych. Zdecydowaliśmy się więc pozbyć się tych rekordów. Po tej Operacji pozostało nam 2140 rekordów. Są one gotowe do dostarczenia do modelu, jednak ich ilość jest poniżej oczekiwań.

2.2.2 rozwiązanie problemu

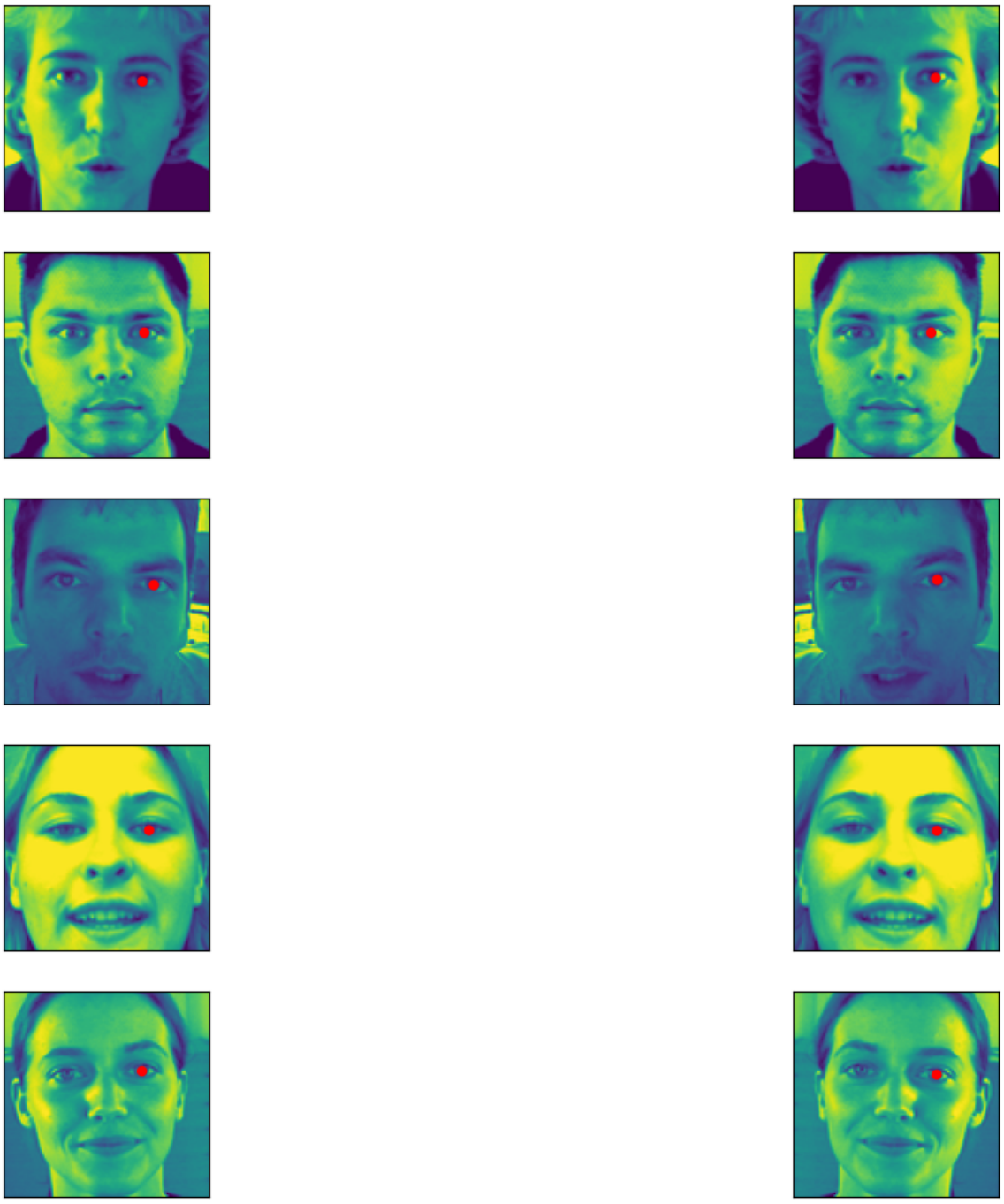
Aby rozwiązać problem małego zbioru danych, postanowiliśmy rozwiązać go, obracając wszystkie obraz i wszystkie cechy lustrzanie odbito względem osi OX. W celu ułatwienia tej operacji, wszystkie współrzędne cech zostały najpierw znormalizowane do przedziału $[-1, 1]$. W wyniku tych operacji otrzymaliśmy nowy zbiór danych. Porównanie odpowiadających sobie zdjęć i cech zostało przedstawione na Rysunku 2.



Rysunek 2: Rysunek przedstawia porównanie zdjęć i cech z wyjściowego zbioru danych z nowym zbiorem otrzymanym poprzez odbicia lustrzane odbicia względem osi OX

Takie operacje rodzą jednak kolejny problem. Mianowicie dla wierszy odbitych, pozycje lewych

cech na zdjęciach, odpowiadają danym opisanym jako prawe cechy i na odwrót. Aby to rozwiązać wystarczy jedynie pozamieniać wartości w odpowiadających kolumnach. Po tej operacji otrzymamy już całkowicie spójne dwa zbiory. dowód tego widać na Rysunku 3



Rysunek 3: Rysunek przedstawia porównania odpowiadających sobie zdjęć z nałożonymi na nie pozycji lewych oczu. jak widać znajdują się one po tej samej stronie niezależnie od tego czy zdjęcie było odbite czy nie

Teraz Wystarczy połączyć oba zbiory w jeden, i otrzymamy zbiór 4280 rekordów gotowych do modelu.

3 Trenowanie modeli

Zgodnie z założeniami, udało nam się zaimplementować dwa modele oraz jeden dodatkowy:

- prostszy konwolucyjnej sieci neuronowej(CNN) w bibliotece Tensorflow
- złożony konwolucyjnej sieci neuronowej(CNN) w bibliotece Tensorflow
- wielowarstowy perceptron(MLP) w bibliotece Pytorch

W wszystkich modelach zdecydowaliśmy się potraktować problem, jako zadania regresji. Naszym wejściem jest wektor rozmiaru 9216(96×96=wymiar wejściowego obrazu), a wyjście to 30-elementowy wektor, reprezentujący współrzędne kolejnych cech twarzy. Wszystkie wykresy oraz fragmenty kodu z tej sekcji, znajdują się w repozytorium projektu w plikach z folderów odpowiednio: Model_tf oraz MLPModel.

3.1 Trenowanie CNN

3.1.1 architektura i zbiór danych

W postszym modelu CNN zdecydowaliśmy się zaimplementować model składający się z 20 warstw pierwsze 10 warstw składa się z 5 warstw Conv2D (o filtrach odpowiednio 32, 64, 128, 256, 512 i metodzie aktywacji ReLU), gdzie po każdej występuje warstwa MaxPooling2D. Następnie umieszczono warstwę Flatten. Ostatnie 9 warstw to 5 warstw dense (o unit równym odpowiednio 1024, 512, 256, 128, 30 i aktywacji ReLU), po każdej prócz ostatniej warstwy dodano warstwę dropout z rate = 0.5. Model złożony nie różnii się zbytnio od modelu prostszego, model ten posiada 5 warstw więcej, które są zduplikowanymi warstwami Conv2D. Implementację tych architektur można zobaczyć odpowiednio na rysunku 4 raz 5.

```
model_1 = Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(96, 96, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(512, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),

    layers.Dense( units: 1024, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense( units: 512, activation='relu'),
    layers.Dropout(0.5),  # Adding dropout for regularization
    layers.Dense( units: 256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense( units: 128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(30)
])
```

Rysunek 4: Implementacja architektury prostszego modelu CNN w bibliotece TensorFlow

```
model_2 = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(96, 96, 1)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(512, (3, 3), activation='relu'),
    layers.Conv2D(512, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),

    layers.Dense( units: 1024, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense( units: 512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense( units: 256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense( units: 128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(30)  # Output layer with 30 neurons
])
```

Rysunek 5: Implementacja architektury złożonego modelu CNN w bibliotece TensorFlow

Za zbiór danych, przyjęliśmy wcześniej przetworzony przez nas zbiór z Kaggla. Następnie, w sposób losowy, wydzieliliśmy z niego 15% na zbiór testowy, a resztę przeznaczyliśmy na trening

3.1.2 Parametry trenowania

Podczas szkolenia modelu, modyfikowaliśmy parametry, do momentu osiągnięcia zadowalających nas wyników na zbiorze testowym. Ostatecznie następujący zbiór parametrów spełnił naszę oczekiwania:

- liczba epok: 40
- rozmiar jednej serii: 16
- learning rate: 0.001
- funkcja straty: MSE(Mean square error)

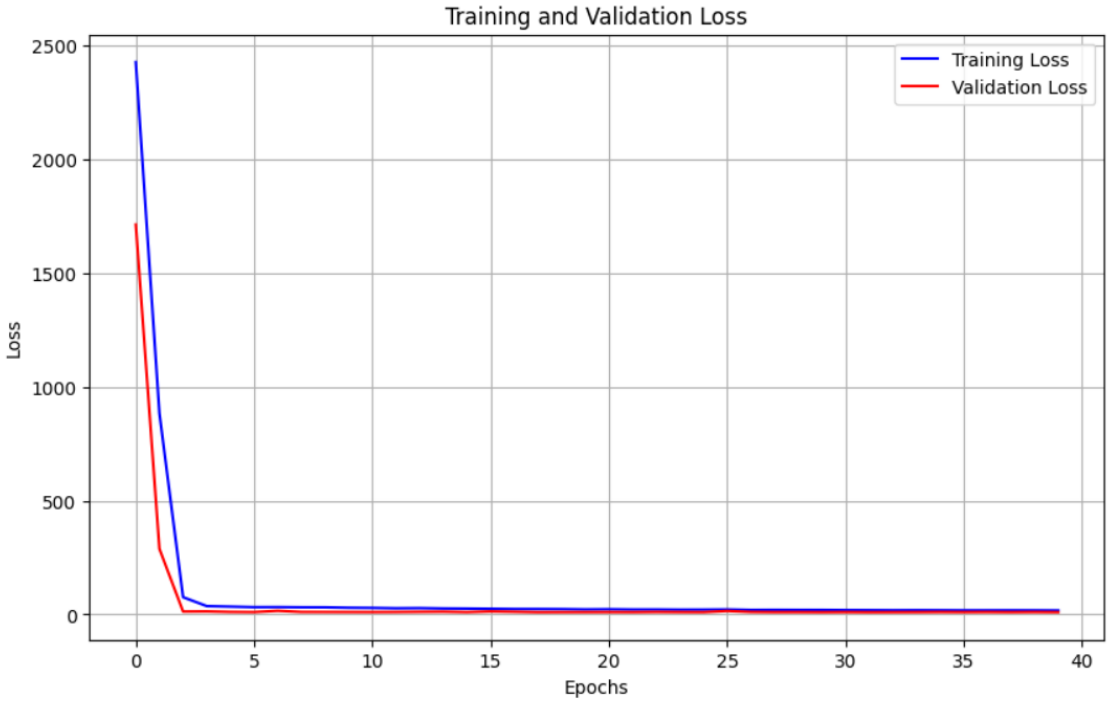
- optymalizator wag modelu: Adam(Adaptive Moment Estimation)

Warty zauważenia jest fakt, że dla większej ilości epok, wartość funkcji straty zaczynała rosnąć. Zapewne było to spowodowane małą ilością neuronów w warstwach ukrytych, w porównaniu z rozmiarem wejścia. Ponieważ jednak udało nam się uzyskać satysfakcjonujący wynik na takiej ilości epok, zdecydowaliśmy się nie modyfikować architektury

3.1.3 trenowanie

Trenowanie modelu przeprowadziliśmy zgodnie ze sztuką, to znaczy za pomocą modelu estymowaliśmy wartości wejścia, wyznaczaliśmy wartość straty od wartości właściwej, a następnie optymalizowaliśmy wagi modelu. Po wytenowaniu modelu wkorzystalimy history zwracane przez metodę model.fit(), aby wykreślić wykres training oraz validation loss. Wykresy te znajdują się na rysunkach 6 oraz 7

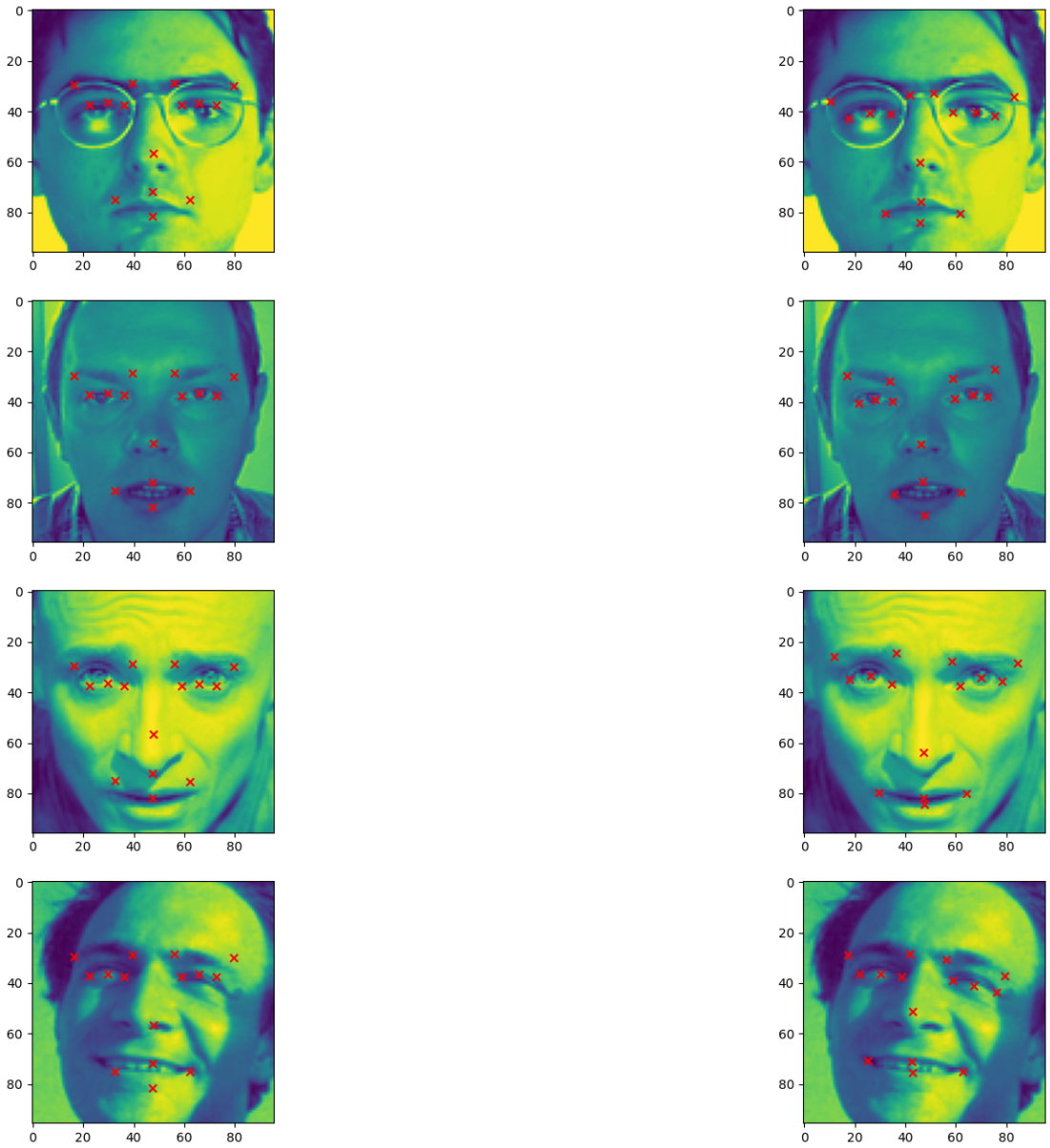
Rysunek 6: Wykres validation oraz training loss w zależności od epoki trenowania pierwszego modelu CNN



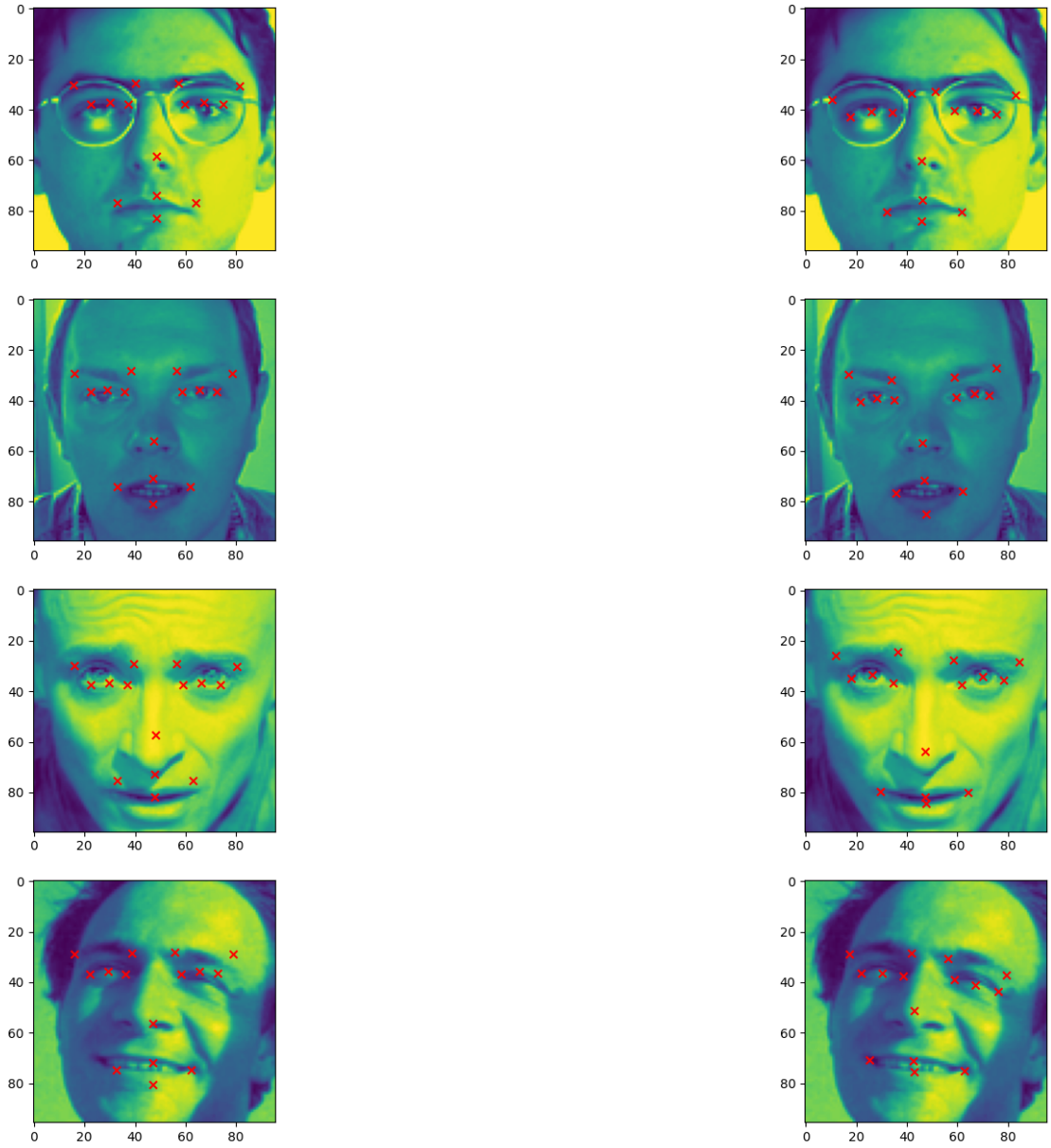
Rysunek 7: Wykres validation oraz training loss w zależności od epoki trenowania drugiego modelu CNN

3.1.4 testowanie

W ramach testu, zdecydowaliśmy się sprawdzić jaką skuteczność posiada nasz model, w przewidzeniu współrzędnych każdej z cech. Każda próbka reprezentowała 30 współrzędnych. Uznawaliśmy że model poprawnie przewidział współrzędną jeżeli odległość od rzeczywistej współrzędnej jest mniejsza niż 3 piksele. Za pomocą tej metryki, pierwszy model uzyskał 72.67% celności na zbiorze testowym, natomiast drugi uzyskał skuteczność 72.82%. Taki wynik dla tych modeli jest dla nas wystarczająco zadowalający.
W celu Wizualizacji celności naszego modelu, Wybraliśmy również 4 próbek ze zbioru testowego, na które nałożyliśmy, na oddzielnych obrazkach, współrzędne rzeczywiste i nasze przewidziane. Zestawienie to przedstawiono na rysunkach: 8 oraz 7



Rysunek 8: Zestawienie wyników modelu CNN1 (lewo), z rzeczywistymi (prawo)



Rysunek 9: Zestawienie wyników modelu CNN2 (lewo), z rzeczywistymi (prawo)

3.2 Trenowanie MLP

3.2.1 architektura i zbiór danych

W modelu MLP zdecydowaliśmy się zaimplementować architekturę składającą się z 2 warstw ukrytych, posiadających po 1036 neuronów. Za funkcję aktywacji na każdej z tych warstw wybraliśmy

ReLU. Aby ograniczać ryzyko overfittingu, staraliśmy się unikać powiększania sieci.

Implementację tej architektury można zobaczyć na rysunku 10

```
class MLP(nn.Module):
    PioRow
    def __init__(self, input_dim, output_dim):
        super(MLP, self).__init__()
        self.l1 = nn.Linear(input_dim, out_features: 1036)
        self.relu1 = nn.ReLU()
        self.l2 = nn.Linear( in_features: 1036, out_features: 1036)
        self.relu2 = nn.ReLU()
        self.l3 = nn.Linear( in_features: 1036,output_dim)
```

Rysunek 10: Implementacja architektury modelu MLP w bibliotece PyTorch

Za zbiór danych przyjęliśmy, wcześniej przetworzony przez nas, zbiór z Kaggla. Jedyną modyfikacją, którą do niego wprowadziliśmy, to normalizacja pikseli obrazka, z przedziału [0,255] do przedziału [0,1]. Następnie, w sposób losowy, wydzieliliśmy z niego 15% na zbiór testowy, a resztę przeznaczyliśmy na trening.

3.2.2 Parametry trenowania

Podczas szkolenia modelu modyfikowaliśmy parametry do momentu osiągnięcia zadowalających nas wyników na zbiorze testowym.

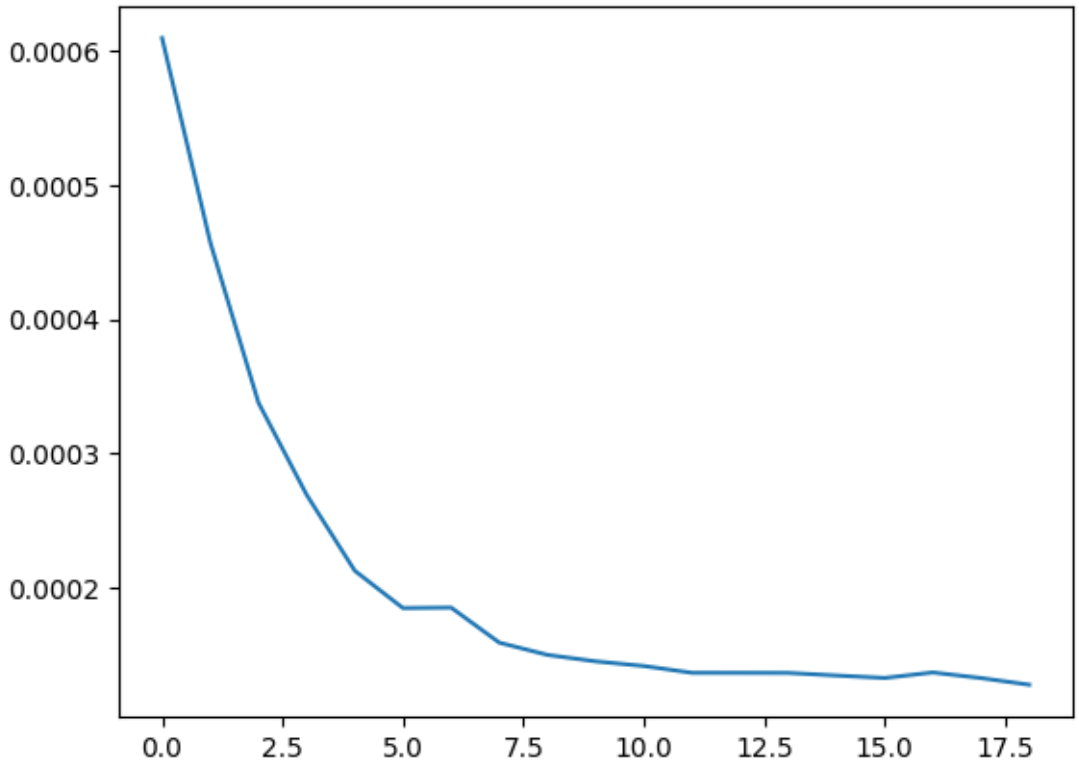
Ostatecznie, następujący zbiór parametrów spełnił nasze oczekiwania:

- liczba epok: 20
- rozmiar jednej serii: 20
- learning rate: 0.005
- funkcja straty: MSE(Mean square error)
- optymalizator wag modelu: Adam(Adaptive Moment Estimation)

Warty zauważenia jest fakt, że dla większej ilości epok wartość funkcji straty zaczynała rosnąć. Podejrzewamy, że było to spowodowane małą ilością neuronów w warstwach ukrytych, w porównaniu z rozmiarem wejścia. Jednak, skoro udało nam się uzyskać satysfakcjonujący wynik na takiej ilości epok, zdecydowaliśmy się nie modyfikować architektury.

3.2.3 trenowanie

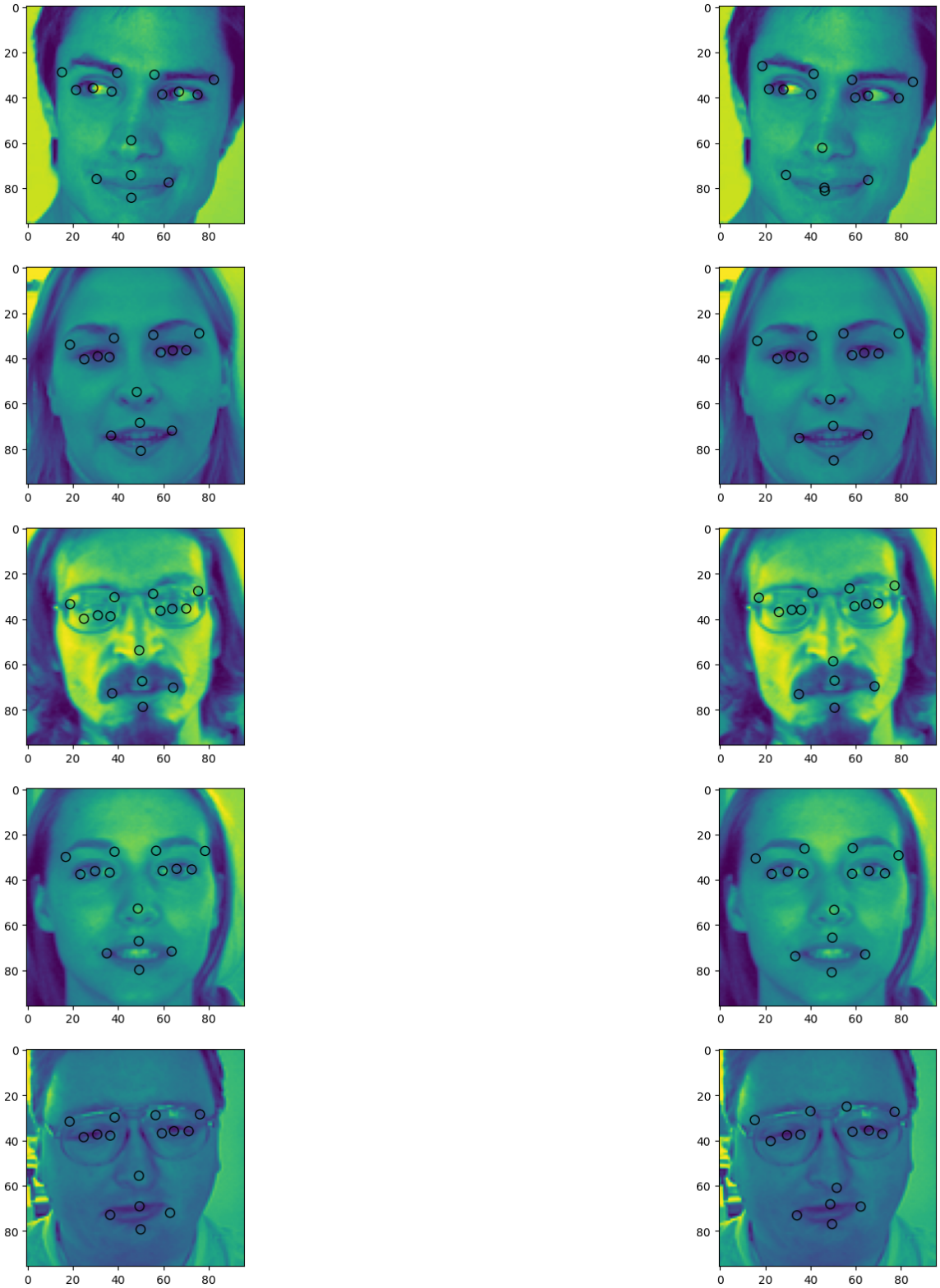
Trenowanie modelu przeprowadziliśmy zgodnie ze sztuką, to znaczy - za pomocą modelu estymowaliśmy wartości wejścia, wyznaczaliśmy wartość straty od wartości właściwej, a następnie optymalizowaliśmy wagi modelu. Podczas trenowania modelu, zapamiętywaliśmy również średnią wartość straty, a następnie wykreśliliśmy jej zależność od epoki. Wykres ten znajduje się na rysunku 11.



Rysunek 11: Wykres średniej wartości straty(loss), w zależności od epoki dla trenowania modelu MLP.

3.2.4 Testowanie

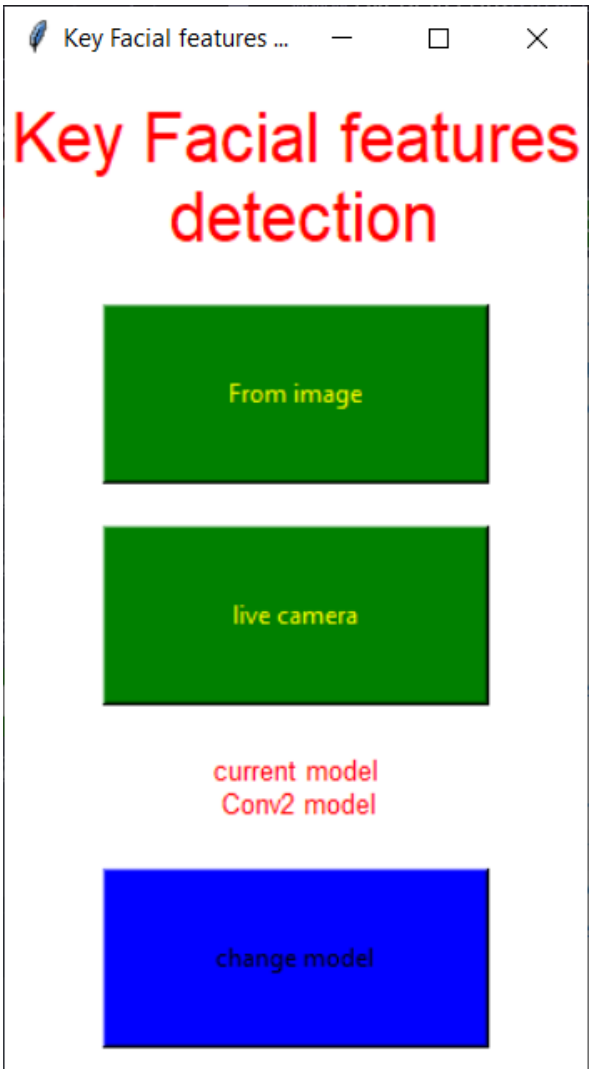
W ramach testu zdecydowaliśmy się sprawdzić, jaką skuteczność posiada nasz model w przewidywaniu współrzędnych każdej z cech. Każda próbka reprezentowała 30 współrzędnych. Uznawaliśmy, że model poprawnie przewidział współrzędną, jeżeli odległość od rzeczywistej współrzędnej była mniejsza niż 2 piksele. Za pomocą tej metryki, model uzyskał 67.53% celności na zbiorze testowym. Taki wynik dla tego modelu jest dla nas bardzo zadowalający.
W celu wizualizacji celności naszego modelu wybraliśmy również 5 próbek ze zbioru testowego, na które nałożyliśmy, na oddzielnych obrazkach, współrzędne rzeczywiste i nasze przewidziane. Zestawienie to przedstawiono na rysunku 12. Jak możemy zauważyć, model na ogół trafia w miejsce zbliżone do oryginalnego.



Rysunek 12: Zestawienie wyników modelu MLP(lewo), z rzeczywistymi (prawo)

4 GUI

W celu wizualizacji naszych modeli, stworzyliśmy prosty interfejs graficzny. Po uruchomieniu głównego skryptu wyświetli się menu główne aplikacji.



Rysunek 13: Wygląd menu głównego

Dostępne są w nim 3 przyciski: *From Image* i *live camera*, których funkcjonalność opisana została w specjalnych sekcjach oraz *change model* która służy do zamiany modelu wykorzystywanego do znajdywania punktów charakterystycznych. Aktualnie wykorzystywany model można odczytać z etykiety znajdującej się nad przyciskiem.

4.1 From image

Po naciśnięciu przycisku, wyświetlony zostanie eksplorator plików, z którego należy wybrać zdjęcie do przetworzenia. Po jego wyborze za pomocą biblioteki OpenCV, zlokalizowane zostaną wszystkie twarze. Następnie za pomocą wybranego modelu zostaną zlokalizowane punkty charakterystyczne, a następnie naniesione i wyświetlone w nowym oknie.

Rysunek 14: Przykład działania aplikacji dla zdjęcia

Aby powrócić do menu głównego, należy nacisnąć dowolny przycisk klawiatury.

4.2 Live camera

po naciśnięciu przycisku live camera, aplikacja zacznie przechwytywać i wyświetlać obraz z kamery urządzenia. Aby wykryć cechy charakterystyczne, należy nacisnąć przycisk „spacja”. Wówczas aktualna klatka kamery zostanie przetworzona identycznie jak z sekcji from image.

Rysunek 15: Caption

Aby powrócić do menu głównego z poziomu kamery, należy nacisnąć przycisk „q”.

5 Podsumowanie

Udało nam się przetworzyć dataset do naszego pomysłu, zaprojektować i wytrenować modele o różnych typach oraz napisać aplikację, do wizualizacji działania naszych projektów.

Wszystkie wstępne założenia zostały spełnione, a część z nich nawet rozszerzona (dodatkowy model CNN, Wizualizacja live oraz ze zdjęcia).