Politechnika Wrocławska

Podstawy Sieci Neuronowych – Rozpoznawanie Gestów Dłoni

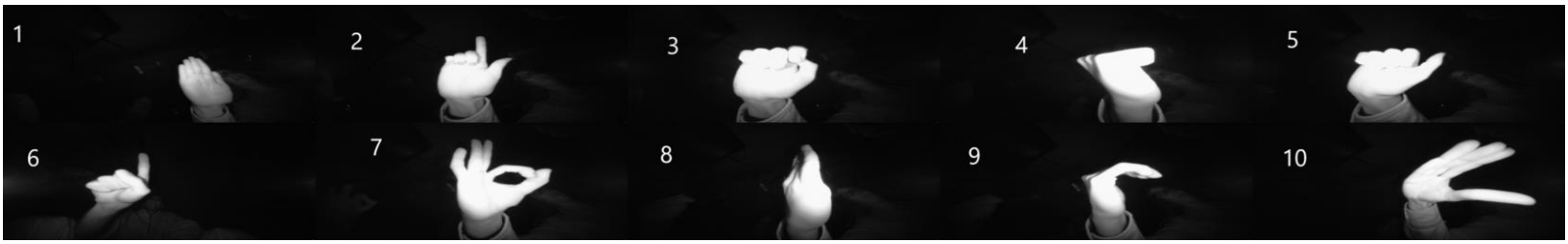
Hubert Grzegorzewski

Wprowadzenie

Celem projektu było zbudowanie i przetestowanie sieci neuronowych której zadaniem jest rozpoznanie i sklasyfikowanie gestów dłoni na jedną z dziesięciu kategorii.

Dane składają się z 20 000 obrazów podzielonych na 10 klas odpowiadających różnym gestom. Każda klasa posiada 2 000 obrazów.

1. Otwarta Dłoń
2. Litera L
3. Pięść
4. Odwrócona pięść
5. Pięść z kciukiem
6. Palec wskazujący
7. “Ok”
8. Obrócona dłoń
9. Litera C
10. Kciuk w dół



Rysunek 1 : *Obraz przedstawiający każdą klase gestu*

Przygotowanie Danych

1. **Normalizacja:** Wartości pikseli przeskalowano do przedziału [0, 1] przy użyciu ImageDataGenerator.
2. **Augmentacja:** Aby zwiększyć różnorodność danych treningowych, zastosowano:
    - Rotacje (±15°)
    - Przesunięcia w pionie i poziomie (±10%)
    - Zoom (±20%)
    - Odbicia poziome.
3. **Podział danych:** Dane podzielono na trzy zbiory:
    - Treningowy (70%)
    - Walidacyjny (30%)
    - Testowy (osobny folder zawierający 300 obrazów na klasę).

Format danych: obrazy w skali szarości o wymiarach 640x240 pikseli zorganizowane w katalogach.

```python
image_size = (640, 240)  # Rozmiar obrazów (640x240 pikseli)
data_dir = 'Data/leapGestRecog'  # Ścieżka do katalogu z obrazami
test_data_dir = 'Data/TEST'

# ImageDataGenerator do skalowania wartości pikseli i podziału na zbiór treningowy/walidacyjny
data_gen = ImageDataGenerator(
    rescale=1./255,  # Normalizacja pikseli do zakresu [0, 1]
    rotation_range=15,
    width_shift_range=0.125,
    height_shift_range=0.125,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # Podział na zbiór walidacyjny (20%)
)

train_data = data_gen.flow_from_directory(
    data_dir,
    target_size=image_size,
    color_mode='grayscale',  # Skala szarości
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_data = data_gen.flow_from_directory(
    data_dir,
    target_size=image_size,
    color_mode='grayscale',
    batch_size=32,
    class_mode='categorical',
    subset='validation'
```

Rysunek 2 : *Przygotowanie danych*

Architektura Sieci

Model 1

- **Warstwy:**
    - Warstwa wejściowa : Zdjęcie w skali szarości 640x240
    - 3 warstwy konwolucyjne (rozmiar filtrów: 3x3)
    - MaxPooling (2x2)
    - Warstwa w pełni połączona (512 neuronów)
    - Dropout (0.5)
    - Warstwa wyjściowa z liczbą neuronów odpowiadająca liczbie klas(10)
- **Funkcje aktywacji:** ReLU dla warstw ukrytych, Softmax dla warstwy wyjściowej.
- **Optymalizator:** Adam (learning rate = 0.001).
- **Liczba epok:** 25.
- **Batch size:** 32.

```python
# Model 1: CNN
model1 = models.Sequential([
    layers.Input(shape=(640, 240, 1)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    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.Flatten(),
    layers.Dense( units: 512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(train_data.num_classes, activation='softmax')
])


# 3. Kompilacja i trening modelu (Model 1)
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
               loss='categorical_crossentropy',
               metrics=['accuracy'])

history1 = model1.fit(
    train_data,
    validation_data=val_data,
    epochs=25,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    validation_steps=val_data.samples // val_data.batch_size
)
```

Rysunek 3 : *Konfiguracja modelu nr 1*

Model 2

- **Warstwy:**
    - 4 warstwy konwolucyjne (rozmiar filtrów: 3x3)
    - MaxPooling (2x2)
    - Warstwa w pełni połączona (512 neuronów)
    - Dropout (0.425)
    - Warstwa wyjściowa z liczbą neuronów odpowiadająca liczbie klas(10)
- **Funkcje aktywacji:** ReLU dla warstw ukrytych, Softmax dla warstwy wyjściowej.
- **Funkcja straty :** Cross Entropy Loss
- **Optymalizator:** Adam (learning rate = 0.00075).
- **Liczba epok:** 40.
- **Batch size:** 32.

```python
# model 2 : Rozbudowana architektura CNN
model2 = models.Sequential([
    layers.Input(shape=(640, 240, 1)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    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.Flatten(),
    layers.Dense( units: 512, activation='relu'),
    layers.Dropout(0.425),
    layers.Dense(train_data.num_classes, activation='softmax')
])


# 3. Kompilacja i trening modelu (Model 2)
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00075),
               loss='categorical_crossentropy',
               metrics=['accuracy'])

history2 = model2.fit(
    train_data,
    validation_data=val_data,
    epochs=40,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    validation_steps=val_data.samples // val_data.batch_size
)
```

Rysunek 4 : *Konfiguracja modelu nr 2*

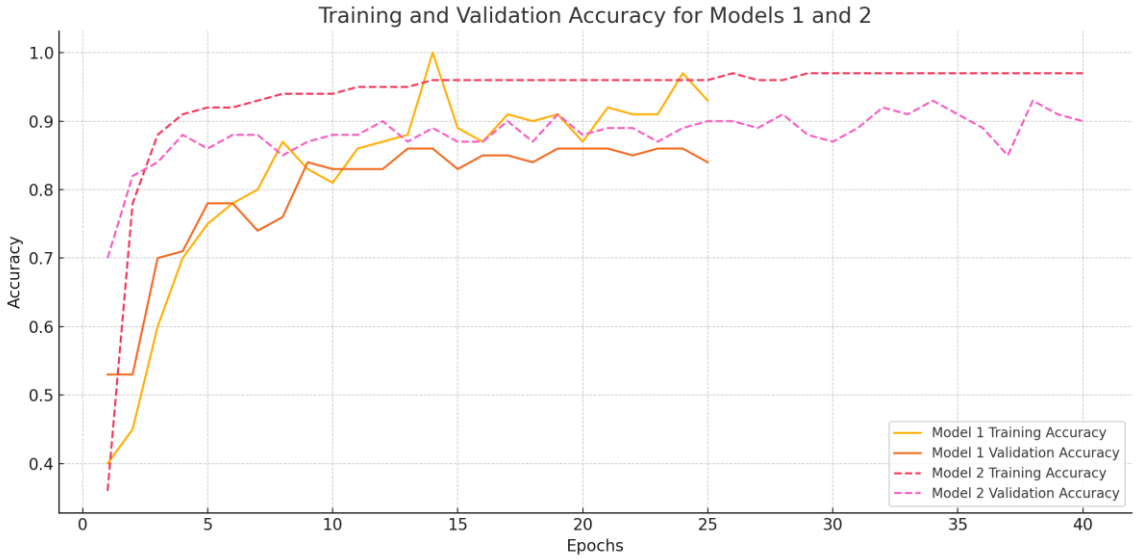
Trenowanie Sieci i Eksperyment

Konfiguracje

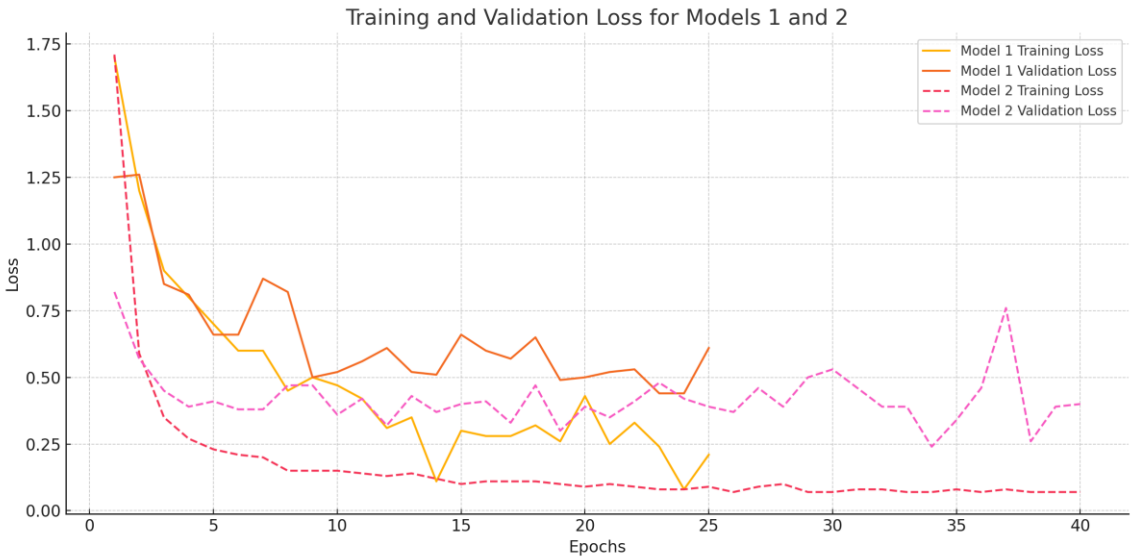
1. Model 1: Podstawowa architektura CNN.
2. Model 2: Rozbudowana architektura CNN

Automatyzacja

Proces treningu został zautomatyzowany przy użyciu TensorFlow. Generowano wykresy strat i dokładności, a modele zapisywano w formacie HDF5.



Rysunek 5 : *Wykres przedstawiający skuteczność rozpoznawania gestów na danych uczących i walidacyjnych dla modelu nr 1 i 2*



Rysunek 6 : *Wykres przedstawiający skuteczność rozpoznawania gestów na danych uczących i walidacyjnych dla modelu nr 1 i 2*

Analiza Wyników

Model 1

- Maksymalna walidacyjna dokładność: 85.7% (epoka 23).
- Straty walidacyjne: 0.44 (epoka 25).
- Wysoka zbieżność między stratami treningowymi i walidacyjnymi.

Model 2

- Maksymalna walidacyjna dokładność: 91.9% (epoka 32).
- Straty walidacyjne: 0.24 (epoka 34).
- Model charakteryzuje się lepszym wygładzeniem krzywych strat.

Raport z klasyfikacji

Model 2 znacząco przewyższa Model 1, osiągając 97% dokładności w porównaniu do 86% Modelu 1. Rozwiązuje problemy z trudnymi klasami, takimi jak "Litera L" (recall wzrósł z 0.07 do 0.73) i "Obrócona dłoń" (recall z 0.50 do 1.00). Jest bardziej niezawodny, lepiej generalizuje i jest zdecydowanie lepszym wyborem. Model 1 ma problemy z rozpoznawaniem niektórych klas i generalizacją.

Raport klasyfikacyjny dla modelu nr 1

Class	Precision	Recall	F1-Score
Otwarta dłoń	1.0	1.0	1.0
Litera L	1.0	0.07	0.12
Pięść	1.0	1.0	1.0
Odwrócona pięść	1.0	1.0	1.0
Pięść z kciukiem	1.0	1.0	1.0
Palec wskazujący	0.41	1.0	0.58
“Ok”	1.0	1.0	1.0
Obrócona dłoń	1.0	0.5	0.67
Litera C	1.0	1.0	1.0
Kciuk w dłoń	1.0	1.0	1.0

Metric	Value
Accuracy	0.86
Weighted Avg Precision	0.94
Weighted Avg Recall	0.86
Weighted Avg F1-Score	0.84

Raport klasyfikacyjny dla modelu nr 2

Class	Precision	Recall	F1-Score
Otwarta dłoń	1.0	1.0	1.0
Litera L	1.0	0.73	0.85
Pięść	0.97	1.0	0.98
Odwrócona pięść	1.0	0.97	0.98
Pięść z kciukiem	1.0	1.0	1.0
Palec wskazujący	0.79	1.0	0.88
“Ok”	1.0	1.0	1.0
Obrócona dłoń	1.0	1.0	1.0
Litera C	1.0	1.0	1.0
Kciuk w dół	1.0	1.0	1.0

Metric	Value
Accuracy	0.97
Ważona średnia Precyzji	0.98
Ważona średnia Recall	0.97
Weighted Avg F1-Score	0.97

Wnioski

- Model 2 osiąga znacznie lepszą dokładność ogólną (97%) w porównaniu do Modelu 1 (86%).
- Model 2 skuteczniej rozpoznaje trudne klasy, takie jak "Litera L" (recall wzrósł z 0.07 do 0.73) i "Obrócona dłoń" (recall wzrósł z 0.50 do 1.00).
- Model 1 wykazuje problemy z generalizacją, co objawia się wyższym validation loss w porównaniu do training loss oraz niestabilnym validation accuracy.
- Model 2 prezentuje stabilny i przewidywalny spadek zarówno training, jak i validation loss, co świadczy o braku przeuczenia.
- Validation accuracy Modelu 2 jest stabilne i zbliżone do training accuracy, co potwierdza jego dobrą zdolność generalizacji.
- Model 1 ma problemy z rozpoznawaniem klasy "Litera L", gdzie większość przykładów tej klasy jest pomijana (niski recall: 0.07).
- Model 2 jest bardziej niezawodny i lepiej dopasowany do danych w porównaniu do Modelu 1, co czyni go bardziej odpowiednim do zastosowania.