

Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej

przy użyciu Konwolucyjnych Sieci
Neuralnych





Cel Projektu

Głównym celem tego projektu jest klasyfikacja chorób na podstawie zdjęć rentgenowskich klatki piersiowej, co ma na celu usprawnienie i przyspieszenie procesu diagnozy.

Cel Projektu

The background is a deep blue with intricate, swirling, marbled patterns that resemble liquid or smoke. On the right side, there is a large, semi-transparent hexagonal graphic composed of several overlapping layers in different shades of blue and white. A black rectangular box is positioned in the upper right area, containing the text 'Cel Projektu' in white.

Cel Projektu

Cel Projektu

Wykorzystujemy do tego zaawansowane technologie, takie jak sieci konwolucyjne, które są częścią głębokiego uczenia się, podgałęzi sztucznej inteligencji.

The background is a deep blue nebula with intricate, swirling patterns of gas and dust, punctuated by numerous small, bright white stars. In the upper right, a solid black rectangular box contains the text 'Cel Projektu' in white. In the lower right, there is a large, stylized graphic composed of several overlapping, semi-transparent blue and white geometric shapes, including a prominent hexagon and various trapezoids, creating a modern, architectural feel.

Cel Projektu

Cel Projektu

Automatyzacja diagnozowania na podstawie obrazów medycznych ma ogromne znaczenie dla medycyny, poprawiając szybkość i skuteczność leczenia.

The background is a deep blue nebula with intricate, swirling patterns of gas and dust, punctuated by numerous small, bright white stars. In the upper right, a solid black rectangular box contains the text 'Cel Projektu' in white. In the lower right, there is a large, stylized graphic composed of several overlapping, semi-transparent blue and white geometric shapes, including a prominent hexagon and various trapezoids, creating a modern, architectural feel.

Cel Projektu

Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej

przy użyciu Konwolucyjnych Sieci
Neuralnych





Opis Datasetu

Dane pochodzą z publicznie dostępnych zbiorów danych zawierających obrazy rentgenowskie klatki piersiowej wraz z etykietami chorób.

Opis Datasetu

Dane pochodzą z publicznie dostępnych zbiorów danych zawierających obrazy rentgenowskie klatki piersiowej wraz z etykietami chorób.

Zestaw danych składa się z tysięcy obrazów, każdy reprezentujący unikalne warunki zdrowotne pacjenta.

Opis Datasetu

Dane pochodzą z publicznie dostępnych zbiorów danych zawierających obrazy rentgenowskie klatki piersiowej wraz z etykietami chorób.

Opis Datasetu

Zestaw danych składa się z tysięcy obrazów, każdy reprezentujący unikalne warunki zdrowotne pacjenta.

Obrazy są klasyfikowane według różnych chorób, takich jak zapalenie płuc, torbiel, guz, itp.

Opis Datasetu

Zestaw danych składa się z tysięcy obrazów, każdy reprezentujący unikalne warunki zdrowotne pacjenta.

Obrazy są klasyfikowane według różnych chorób, takich jak zapalenie płuc, torbiel, guz, itp.

Opis Datasetu

Obrazy są klasyfikowane według różnych chorób, takich jak zapalenie płuc, torbiel, guz, itp.




Opis Datasetu

Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej


przy użyciu Konwolucyjnych Sieci
Neuralnych





Każdy obraz jest przekształcany za pomocą procesu przetwarzania wstępnego: zmiana rozmiaru do ustalonego wymiaru, konwersja do skali szarości, i normalizacja.

**Przetwarzanie
Wstępne**




Etykiety chorób są kodowane na wektory przy użyciu one-hot encoding.

Kod odpowiedzialny za wstępne przetworzenie

```
train_transform = transforms.Compose([
    transforms.Resize((1024, 1024)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])


val_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

test_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Każdy obraz jest przekształcany za pomocą procesu przetwarzania wstępnego: zmiana rozmiaru do ustalonego wymiaru, konwersja do skali szarości, i normalizacja.

**Przetwarzanie
Wstępne**



Etykiety chorób są kodowane na wektory przy użyciu one-hot encoding.

Klasa XrayDataset jest odpowiedzialna za tworzenie niestandardowego zbioru danych dla zadania klasyfikacji rentgenowskich obrazów. Działa jako interfejs między danymi (obrazami rentgenowskimi i etykietami) a modelem.

Kod odpowiedzialny za
one-hot encoding


```
class XrayDataset(Dataset):
    def __init__(self, df, root_dir, transform=None):
        self.labels_frame = df['Finding Labels'].str.get_dummies('|')
        self.image_frame = df['Image Index']
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.image_frame)

    def __getitem__(self, idx):
        img_name = os.path.join(self.root_dir, self.image_frame.iloc[idx])
        image = Image.open(img_name)
        labels = self.labels_frame.iloc[idx].values.astype(np.float32)


        if self.transform:
            image = self.transform(image)

        return image, labels
```

Każdy obraz jest przekształcany za pomocą procesu przetwarzania wstępnego: zmiana rozmiaru do ustalonego wymiaru, konwersja do skali szarości, i normalizacja.

**Przetwarzanie
Wstępne**



Etykiety chorób są kodowane na wektory przy użyciu one-hot encoding.

Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej

przy użyciu Konwolucyjnych Sieci
Neuralnych



Architektura Sieci Neuronalnej

Model


```
model = EfficientNet.from_name('efficientnet-b6')  
num_fts = model._fc.in_features  
model._fc = nn.Linear(num_fts, len(train_dataset[0][1]))
```

Optymalizator

```
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```


Unikanie
przetrenowania

```
if epoch_val_loss < best_val_loss:  
    best_val_loss = epoch_val_loss  
    early_stop_counter = 0  
else:  
    early_stop_counter += 1  
  
if early_stop_counter >= patience:  
    print("Early Stopping: No improvement in validation loss. Training stopped.")  
    break
```


A blue 3D hexagonal icon with a white dot in the center, representing the ResNet-152 model.

ResNet-152

ResNet (Residual Neural Network) to model, który wprowadza koncepcję połączeń skrótu, które pomagają przeciwdziałać problemowi zanikającego gradientu w głębokich sieciach. ResNet-152 jest jednym z większych wariantów, z 152 warstwami. Dzięki swojej strukturze ResNet jest w stanie efektywnie uczyć się z dużej liczby warstw bez utraty wydajności.

A blue 3D hexagonal icon with a white dot in the center, representing the EfficientNet-B6 model.

EfficientNet-B6

EfficientNet to seria modeli zaprojektowanych z myślą o skali. Zamiast dodawać więcej warstw, jak w przypadku ResNet, EfficientNet skupia się na równoczesnym zwiększaniu głębokości (liczby warstw), szerokości (liczby kanałów) i rozdzielczości (rozmiar wejścia). EfficientNet-B6 to szósta iteracja tej serii. Zasadniczo EfficientNet dąży do osiągnięcia wyższej wydajności przy niższym zużyciu zasobów.

Model



ResNet-152

Wady i zalety

Wady

ResNet-152

Wady i zalety

Wady

Złożoność: ResNet-152, mając 152 warstwy, jest dosyć złożonym modelem, który może wymagać dużo mocy obliczeniowej i pamięci.

Czas treningu: Ze względu na swoją złożoność, ResNet-152 może potrzebować więcej czasu na trening w porównaniu do mniejszych modeli.

Optymalizacja: Mimo że ResNet radzi sobie z problemem zanikającego gradientu, może być trudno zoptymalizować sieć do osiągnięcia najlepszych możliwych wyników, szczególnie gdy model staje się coraz głębszy.

Potrzeba dużych zbiorów danych: Podobnie jak inne głębokie sieci neuronowe, ResNet-152 może potrzebować dużych, dobrze oznaczonych zbiorów danych do skutecznego uczenia się.

ResNet-152

Wady i zalety

Złożoność: ResNet-152, mając 152 warstwy, jest dosyć złożonym modelem, który może wymagać dużo mocy obliczeniowej i pamięci.

Czas treningu: Ze względu na swoją złożoność, ResNet-152 może potrzebować więcej czasu na trening w porównaniu do mniejszych modeli.

Optymalizacja: Mimo że ResNet radzi sobie z problemem zanikającego gradientu, może być trudno zoptymalizować sieć do osiągnięcia najlepszych możliwych wyników, szczególnie gdy model staje się coraz głębszy.

Potrzeba dużych zbiorów danych: Podobnie jak inne głębokie sieci neuronowe, ResNet-152 może potrzebować dużych, dobrze oznaczonych zbiorów danych do skutecznego uczenia się.

ResNet-152

Wady i zalety



ResNet-152

Wady i zalety

Zalety

ResNet-152

Wady i zalety

Zalety

Skuteczność uczenia: ResNet, wprowadzając pojęcie połączeń skrótu (lub "residual connections"), skutecznie radzi sobie z problemem zanikającego gradientu, który jest typowy dla głębokich sieci neuronowych.

Skalowalność: Architektura ResNet jest skalowalna, co oznacza, że można ją łatwo rozszerzyć na większą liczbę warstw bez utraty wydajności.

Wszechstronność: ResNet może być używany do różnorodnych zadań, takich jak klasyfikacja obrazów, detekcja obiektów i segmentacja.

ResNet-152

Wady i zalety

Skuteczność uczenia: ResNet, wprowadzając pojęcie połączeń skrótu (lub "residual connections"), skutecznie radzi sobie z problemem zanikającego gradientu, który jest typowy dla głębokich sieci neuronowych.

Skalowalność: Architektura ResNet jest skalowalna, co oznacza, że można ją łatwo rozszerzyć na większą liczbę warstw bez utraty wydajności.

Wszechstronność: ResNet może być używany do różnorodnych zadań, takich jak klasyfikacja obrazów, detekcja obiektów i segmentacja.


ResNet-152

Wady i zalety




ResNet-152

Wady i zalety

A blue 3D hexagonal icon with a white dot in the center, representing the ResNet-152 model.

ResNet-152

ResNet (Residual Neural Network) to model, który wprowadza koncepcję połączeń skrótu, które pomagają przeciwdziałać problemowi zanikającego gradientu w głębokich sieciach. ResNet-152 jest jednym z większych wariantów, z 152 warstwami. Dzięki swojej strukturze ResNet jest w stanie efektywnie uczyć się z dużej liczby warstw bez utraty wydajności.

A blue 3D hexagonal icon with a white dot in the center, representing the EfficientNet-B6 model.

EfficientNet-B6

EfficientNet to seria modeli zaprojektowanych z myślą o skali. Zamiast dodawać więcej warstw, jak w przypadku ResNet, EfficientNet skupia się na równoczesnym zwiększaniu głębokości (liczby warstw), szerokości (liczby kanałów) i rozdzielczości (rozmiar wejścia). EfficientNet-B6 to szósta iteracja tej serii. Zasadniczo EfficientNet dąży do osiągnięcia wyższej wydajności przy niższym zużyciu zasobów.

Model

EfficientNet-B6

Wady i zalety

Wady

EfficientNet-B6

Wady i zalety

Wady

Złożoność: EfficientNet-B6, będąc jednym z większych wariantów EfficientNet, jest dość złożonym modelem, który może wymagać dużo mocy obliczeniowej i pamięci.

Czas treningu: Podobnie jak w przypadku ResNet-152, trening EfficientNet-B6 może zająć więcej czasu, zwłaszcza gdy używamy dużych zbiorów danych.

Potrzeba dużych zbiorów danych: Aby w pełni wykorzystać potencjał EfficientNet-B6, może być konieczne użycie dużych, dobrze oznaczonych zbiorów danych.

Trudność w dostosowaniu: Ze względu na złożoność architektury, EfficientNet może być trudniejszy do dostosowania i zoptymalizowania w porównaniu do prostszych modeli.

EfficientNet-B6

Wady i zalety

Wady

EfficientNet-B6

Wady i zalety



EfficientNet-B6

Wady i zalety

Zalety

EfficientNet-B6

Wady i zalety

Zalety

Skuteczność: EfficientNets osiąga wyższe wyniki na benchmarkach, takich jak ImageNet, w porównaniu do wielu innych modeli, takich jak ResNets czy DenseNets.

Efektywność: Architektura EfficientNet została zaprojektowana tak, aby zwiększać jednocześnie głębokość (liczbę warstw), szerokość (liczbę kanałów) i rozdzielczość (rozmiar wejścia), co prowadzi do większej efektywności niż tradycyjne podejście do skalowania sieci.

Skalowalność: Tak jak ResNets, EfficientNets są skalowalne, ale z większą efektywnością. Można je łatwo skalować do większych rozmiarów, co może prowadzić do jeszcze lepszej wydajności.

EfficientNet-B6

Wady i zalety

Zalety


EfficientNet-B6

Wady i zalety




EfficientNet-B6

Wady i zalety

A blue 3D hexagonal icon with a white dot in the center, representing the ResNet-152 model.

ResNet-152

ResNet (Residual Neural Network) to model, który wprowadza koncepcję połączeń skrótu, które pomagają przeciwdziałać problemowi zanikającego gradientu w głębokich sieciach. ResNet-152 jest jednym z większych wariantów, z 152 warstwami. Dzięki swojej strukturze ResNet jest w stanie efektywnie uczyć się z dużej liczby warstw bez utraty wydajności.

A blue 3D hexagonal icon with a white dot in the center, representing the EfficientNet-B6 model.

EfficientNet-B6

EfficientNet to seria modeli zaprojektowanych z myślą o skali. Zamiast dodawać więcej warstw, jak w przypadku ResNet, EfficientNet skupia się na równoczesnym zwiększaniu głębokości (liczby warstw), szerokości (liczby kanałów) i rozdzielczości (rozmiar wejścia). EfficientNet-B6 to szósta iteracja tej serii. Zasadniczo EfficientNet dąży do osiągnięcia wyższej wydajności przy niższym zużyciu zasobów.

Model

Architektura Sieci Neuronalnej

Model

```
model = EfficientNet.from_name('efficientnet-b6')  
num_fts = model._fc.in_features  
model._fc = nn.Linear(num_fts, len(train_dataset[0][1]))
```

Optymalizator

```
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```

Unikanie
przetrenowania

```
if epoch_val_loss < best_val_loss:  
    best_val_loss = epoch_val_loss  
    early_stop_counter = 0  
else:  
    early_stop_counter += 1  
  
if early_stop_counter >= patience:  
    print("Early Stopping: No improvement in validation loss. Training stopped.")  
    break
```




SGD

SGD jest jednym z najprostszych algorytmów optymalizacji. Działa on poprzez aktualizację parametrów modelu po każdej iteracji, minimalizując błąd wyjścia. W każdej iteracji używa losowo wybranego podzbioru danych treningowych (zwanego mini-batch) do obliczenia gradientu funkcji błędu. Niestety, SGD ma tendencję do "wędrowania" wzdłuż dolin, co powoduje, że może być trudne osiągnięcie optymalnego punktu. Może także mieć problemy ze zbieganiem, jeśli funkcja błędu jest bardzo nieregularna.



ADAM

ADAM jest bardziej zaawansowanym algorytmem optymalizacji, który jest skonstruowany tak, aby radzić sobie z niektórymi problemami SGD. ADAM używa adaptacyjnych współczynników uczenia się dla różnych parametrów, co oznacza, że niektóre parametry mogą być aktualizowane szybciej, a inne wolniej. ADAM zazwyczaj zbiega szybciej i jest bardziej stabilny niż SGD. Ponadto, ADAM jest mniej wrażliwy na wybór początkowego współczynnika uczenia się.

Optymalizator





SGD

Wady i zalety

Wady

SGD

Wady i zalety

Wady

Wędrowanie: SGD ma tendencję do "wędrowania" wzdłuż dolin, co sprawia, że może być trudne osiągnięcie optymalnego punktu.

Czułość na hiperparametry: SGD jest bardzo wrażliwy na dobór odpowiedniego współczynnika uczenia się. Zbyt duży może prowadzić do niestabilnych wyników, zbyt mały może spowodować bardzo wolne zbieganie.

Brak adaptacyjnej stopy uczenia się: SGD aktualizuje wszystkie wagi jednym i tym samym współczynnikiem uczenia się, co może nie być optymalne, jeśli niektóre wagi powinny być aktualizowane szybciej, a inne wolniej.

Problemy ze zbieganiem: SGD może mieć problemy ze zbieganiem, jeśli funkcja błędu jest bardzo nieregularna lub zawiera wiele minimów lokalnych.

SGD

Wady i zalety

Wady

SGD

Wady i zalety



SGD

Wady i zalety

Zalety

SGD

Wady i zalety

Zalety

Prostota: SGD jest relatywnie prosty do zrozumienia, zaimplementowania i dostosowania. Algorytm jest prosty, co czyni go doskonałym wyborem dla osób rozpoczynających pracę z uczeniem maszynowym.

Skalowalność: SGD jest efektywny obliczeniowo, dzięki czemu jest dobrym wyborem dla dużych zestawów danych.

Efektywność pamięciowa: SGD nie wymaga przechowywania wszystkich danych treningowych w pamięci na raz, co czyni go bardziej efektywnym pamięciowo w porównaniu do algorytmów optymalizacji działających na pełnym zestawie danych.

Właściwości statystyczne: Ze względu na swoją stochastyczną naturę, SGD ma tendencję do "ucieczki" z płytkich minimów lokalnych, które mogą utrudniać proces optymalizacji.

SGD

Wady i zalety

Zalety

SGD

Wady i zalety



SGD

Wady i zalety



SGD

SGD jest jednym z najprostszych algorytmów optymalizacji. Działa on poprzez aktualizację parametrów modelu po każdej iteracji, minimalizując błąd wyjścia. W każdej iteracji używa losowo wybranego podzbioru danych treningowych (zwanego mini-batch) do obliczenia gradientu funkcji błędu. Niestety, SGD ma tendencję do "wędrowania" wzdłuż dolin, co powoduje, że może być trudne osiągnięcie optymalnego punktu. Może także mieć problemy ze zbieganiem, jeśli funkcja błędu jest bardzo nieregularna.



ADAM

ADAM jest bardziej zaawansowanym algorytmem optymalizacji, który jest skonstruowany tak, aby radzić sobie z niektórymi problemami SGD. ADAM używa adaptacyjnych współczynników uczenia się dla różnych parametrów, co oznacza, że niektóre parametry mogą być aktualizowane szybciej, a inne wolniej. ADAM zazwyczaj zbiega szybciej i jest bardziej stabilny niż SGD. Ponadto, ADAM jest mniej wrażliwy na wybór początkowego współczynnika uczenia się.

Optymalizator





ADAM

Wady i zalety

Wady

ADAM

Wady i zalety

Wady

Złożoność: ADAM jest bardziej złożony od SGD pod względem matematycznym i implementacyjnym.

Wielkość pamięci: ADAM przechowuje oszacowania momentów pierwszego i drugiego rzędu dla każdego parametru, co zwiększa wymagania pamięci.

Bias korekta: Choć bias korekta używana w ADAM pomaga stabilizować estymacje momentów, niektóre badania sugerują, że może to prowadzić do nieoptymalnych wyników w niektórych sytuacjach.

Problem z generalizacją: Niektóre badania wykazały, że mimo szybkiego zbiegania, ADAM może mieć problem z generalizacją na niektórych zestawach danych w porównaniu do innych optymalizatorów, takich jak SGD.

ADAM

Wady i zalety

Wady

ADAM

Wady i zalety



ADAM

Wady i zalety

Zalety

ADAM

Wady i zalety

Zalety

Adaptacyjny współczynnik uczenia się: ADAM automatycznie dostosowuje współczynnik uczenia się dla różnych parametrów, co oznacza, że niektóre parametry mogą być aktualizowane szybciej, a inne wolniej. Dzięki temu jest bardziej elastyczny i skuteczny w różnych scenariuszach.

Elastyczność: ADAM działa dobrze z małymi, dużymi i / lub rzadkimi danymi. Jest skuteczny dla różnego rodzaju funkcji błędu i architektur modeli.

Szybkość zbiegania: ADAM zazwyczaj zbiega szybciej niż wiele innych algorytmów optymalizacji, takich jak SGD.

Stabilność: ADAM jest mniej wrażliwy na zmiany współczynnika uczenia się, co czyni go bardziej stabilnym w praktyce.

ADAM

Wady i zalety

Zalety

ADAM

Wady i zalety



ADAM

Wady i zalety



SGD

SGD jest jednym z najprostszych algorytmów optymalizacji. Działa on poprzez aktualizację parametrów modelu po każdej iteracji, minimalizując błąd wyjścia. W każdej iteracji używa losowo wybranego podzbioru danych treningowych (zwanego mini-batch) do obliczenia gradientu funkcji błędu. Niestety, SGD ma tendencję do "wędrowania" wzdłuż dolin, co powoduje, że może być trudne osiągnięcie optymalnego punktu. Może także mieć problemy ze zbieganiem, jeśli funkcja błędu jest bardzo nieregularna.



ADAM

ADAM jest bardziej zaawansowanym algorytmem optymalizacji, który jest skonstruowany tak, aby radzić sobie z niektórymi problemami SGD. ADAM używa adaptacyjnych współczynników uczenia się dla różnych parametrów, co oznacza, że niektóre parametry mogą być aktualizowane szybciej, a inne wolniej. ADAM zazwyczaj zbiega szybciej i jest bardziej stabilny niż SGD. Ponadto, ADAM jest mniej wrażliwy na wybór początkowego współczynnika uczenia się.

Optymalizator



Architektura Sieci Neuronalnej

Model

```
model = EfficientNet.from_name('efficientnet-b6')  
num_fts = model._fc.in_features  
model._fc = nn.Linear(num_fts, len(train_dataset[0][1]))
```

Optymalizator

```
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```

Unikanie
przetrenowania

```
if epoch_val_loss < best_val_loss:  
    best_val_loss = epoch_val_loss  
    early_stop_counter = 0  
else:  
    early_stop_counter += 1  
  
if early_stop_counter >= patience:  
    print("Early Stopping: No improvement in validation loss. Training stopped.")  
    break
```


Co to ?

Early-early stopping jest stosowany w celu uniknięcia przeuczenia modelu, poprawy zdolności do generalizacji oraz zmniejszenia czasu i zasobów potrzebnych do uczenia. Dzięki tej technice możemy wybrać optymalny punkt zatrzymania uczenia modelu, który zapewnia najlepszą wydajność na danych nieznanych.

Early-stopping

Architektura Sieci Neuronalnej

Model

```
model = EfficientNet.from_name('efficientnet-b6')  
num_ftrs = model._fc.in_features  
model._fc = nn.Linear(num_ftrs, len(train_dataset[0][1]))
```

Optymalizator

```
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```

Unikanie
przetrenowania

```
if epoch_val_loss < best_val_loss:  
    best_val_loss = epoch_val_loss  
    early_stop_counter = 0  
else:  
    early_stop_counter += 1  
  
if early_stop_counter >= patience:  
    print("Early Stopping: No improvement in validation loss. Training stopped.")  
    break
```


Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej

przy użyciu Konwolucyjnych Sieci
Neuralnych



Metryki

Prezentacja
Wyników

Wyniki

Accuracy

Loss

Precision

Recall

F1

Metryki

Accuracy

To jest podstawowa metryka, która jest używana do oceny modeli klasyfikacji. Accuracy definiuje stosunek prawidłowo przewidzianych obserwacji do całkowitej liczby obserwacji. Czyli, jest to procent prawidłowych prognoz.

Accuracy

Loss

Precision

Recall

F1

Metryki

Loss

Loss (inaczej koszt) to funkcja, którą chcemy zminimalizować podczas trenowania modelu. W kontekście uczenia maszynowego, loss jest sumą błędów dla każdego przykładu treningowego. Im niższa wartość loss, tym lepszy model.

Accuracy

Loss

Precision

Recall

F1

Metryki

Precision

Precision (precyzja) to stosunek prawidłowo przewidzianych pozytywnych obserwacji do całkowitej liczby przewidzianych pozytywnych obserwacji. Jest to miara, jak wiele z naszych pozytywnych prognoz jest faktycznie prawdziwych.

Accuracy

Loss

Precision

Recall

F1

Metryki

Recall

Recall (czułość) to stosunek prawidłowo przewidzianych pozytywnych obserwacji do wszystkich obserwacji w rzeczywistej klasy pozytywnej. Innymi słowy, jest to miara, jak wiele z rzeczywistych pozytywnych prognozowaliśmy poprawnie.

Accuracy

Loss

Precision

Recall

F1

Metryki

F1 Score

F1 Score to harmoniczna średnia z Precision i Recall. Przyjmuje wartości od 0 do 1, gdzie 1 oznacza doskonałą precyzję i czułość. Ta metryka jest użyteczna, gdy mamy nierównomierny rozkład klas, ponieważ bierze pod uwagę zarówno fałszywe pozytywne, jak i fałszywe negatywne.

Accuracy

Loss

Precision

Recall

F1

Metryki

Metryki

Prezentacja
Wyników

Wyniki

Trening

Prezentacja Wyników

```
C:\Users\piotr\Desktop\PPY-projekt>python main.py
Using 1 GPU for training.
True
11.8
1
NVIDIA GeForce RTX 3080 Ti
Epoch 1/50
-----
Training: 100%[██████████] 69210/69210 [4:52:05<00:00, 3.95it/s, acc=93, f1=0.532, loss=0.201, precision=0.62, recall=0.532]
Training Loss: 0.2008, Training Accuracy: 92.97%, Training F1: 53.22%
Validation: 100%[██████████] 17305/17305 [24:35<00:00, 11.72it/s, acc=93, f1=0.583, loss=0.201, prec=0.563]
Validation Loss: 0.2008, Validation Accuracy: 93.03%, Validation Precision: 58.27%, Validation F1: 58.27%
Epoch 2/50
-----
Training: 87%[██████████] 39290/69210 [2:43:41<2:00:49, 4.13it/s, acc=93, f1=0.54, loss=0.2, precision=0.617, recall=0.54]
```

Test

```
File: 00000013_014.png, Predicted classes: ['No Finding'], True classes: ['No Finding']
File: 00000013_015.png, Predicted classes: ['No Finding'], True classes: ['No Finding']
File: 00000013_016.png, Predicted classes: ['No Finding'], True classes: ['No Finding']
File: 00000013_017.png, Predicted classes: ['No Finding'], True classes: ['No Finding']
```


Metryki

Prezentacja
Wyników

Wyniki

Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej

przy użyciu Konwolucyjnych Sieci
Neuralnych



Wnioski i dalsze kierunki

Mój projekt pokazuje, że zastosowanie konwolucyjnych sieci neuronowych w analizie obrazów medycznych ma duże możliwości.

W przyszłości projekt może być rozwinięty poprzez dodanie więcej chorób do klasyfikacji, poprawę dokładności modelu, czy zastosowanie innych technik uczenia maszynowego.

Klasyfikacja Obrazów Rentgenowskich Klatki Piersiowej

przy użyciu Konwolucyjnych Sieci
Neuralnych

