Akademia Górniczo-Hutnicza im. Stanisława Staszica

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

KLASYFIKACJA ZMIAN CHOROBOWYCH WYWOŁANYCH PRZEZ ZAPALENIE PŁUC NA OBRAZACH RENTGENOWSKICH KLATKI PIERSIOWEJ

Projekt realizowany w ramach przedmiotu
Techniki Obrazowania Medycznego

WYKONAŁY:

Wiktoria Kowalska
Weronika Patuszyńska
Gabriela Piwar
Olivia Ślusarz

III rok Inżynieria Biomedyczna

Kraków, czerwiec 2022

Spis treści:

1. Wprowadzenie

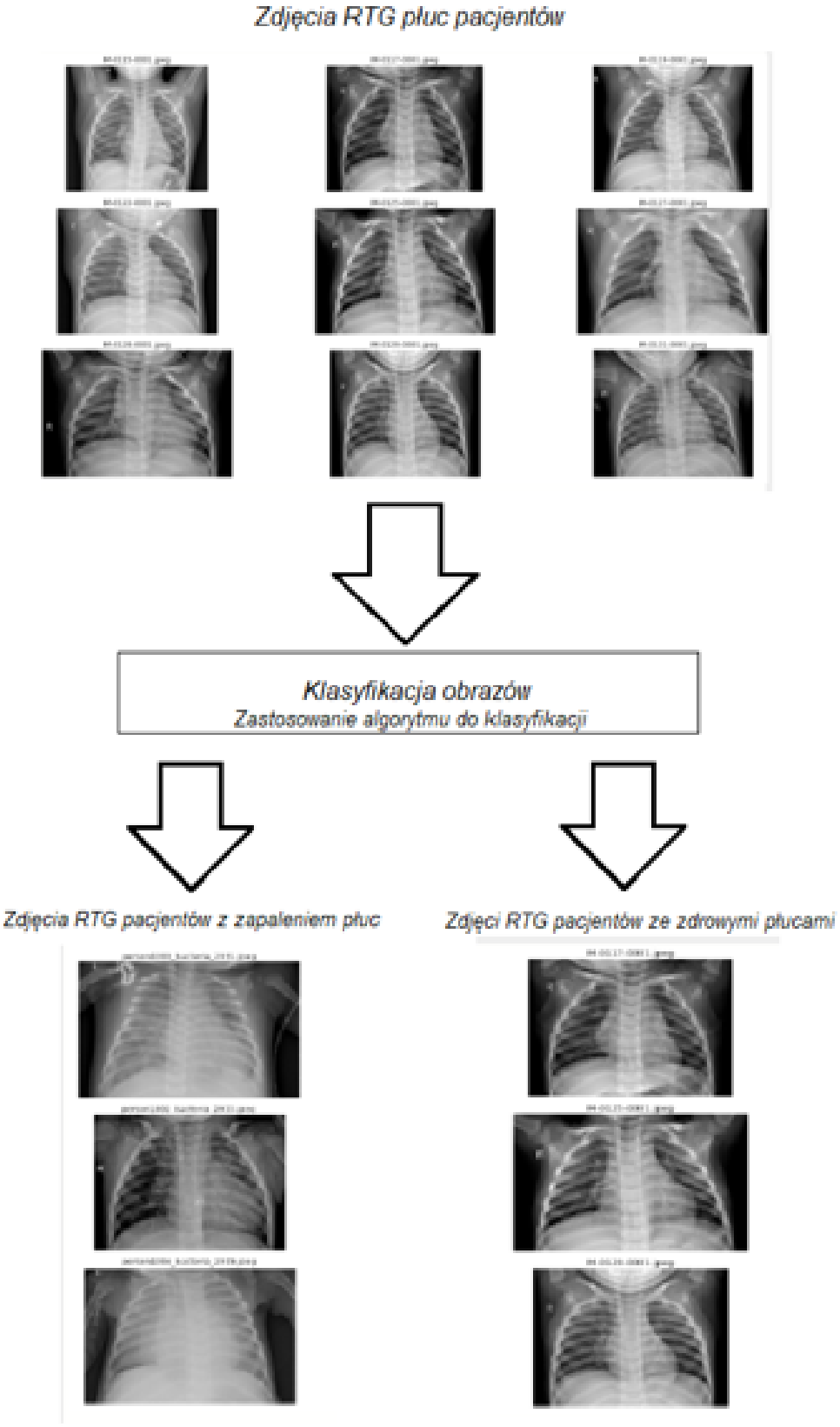
1.1. Temat projektu.

Klasyfikacja zmian chorobowych wywołanych przez zapalenie płuc na obrazach rentgenowskich klatki piersiowej.

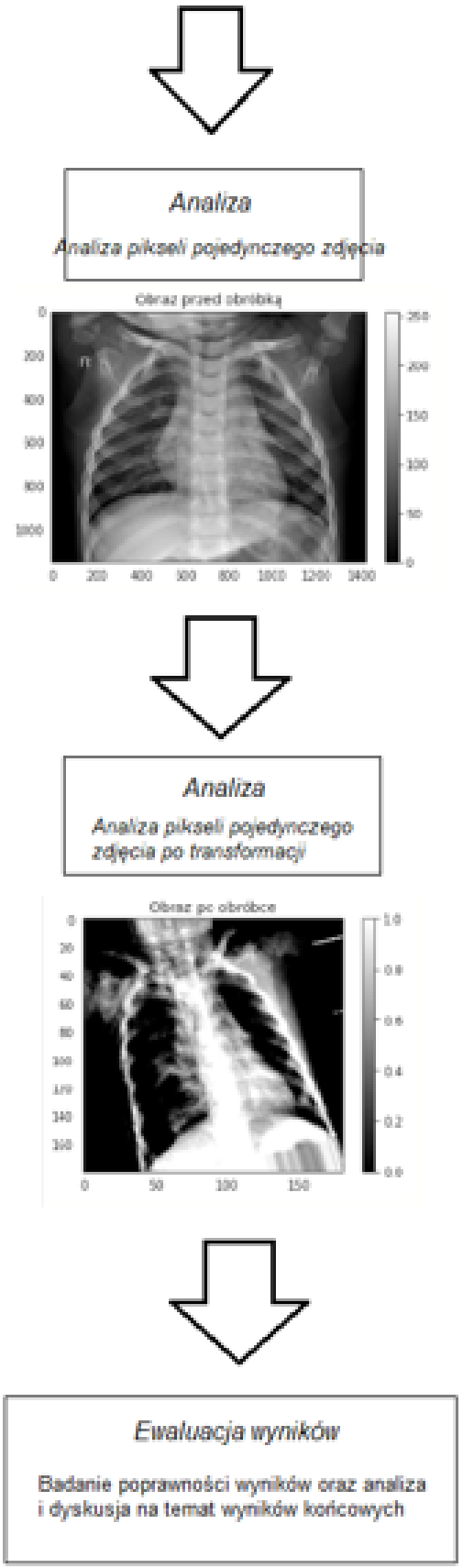
1.2. Cel i założenia projektu.

Celem projektu było zapoznanie się z zagadnieniem klasyfikacji obrazów medycznych oraz implementacja algorytmu do rozpoznawania zdjęć RTG klatki piersiowej charakterystycznych dla osób zdrowych i chorych.. Zrealizowano to za pomocą różnych sieci konwolucyjnych, po czym porównano ich działanie na podstawie różnych metod ewaluacji.

1.3. Abstrakt graficzny.



Rys. 1.3.1 Abstrakt graficzny projektu.



Rys. 1.3.2 Abstrakt graficzny projektu.

1.4. Streszczenie.

W poniżej przedstawionym projekcie dotyczącym klasyfikacji zmian chorobowych wywołanych przez zapalenie płuc na obrazach rentgenowskich klatki piersiowej przedstawiono cel i założenia projektu, a za pomocą abstraktu graficznego zobrazowano jego przebieg. Ponadto zawarto wstęp teoretyczny, który pokrótce scharakteryzował zapalenie płuc, jego najczęściej występujące odmiany i grupy najbardziej narażone na zachorowanie. Dodatkowo opisano czym są sieci konwolucyjne oraz zawarto spis skrótów używanych w projekcie. W projekcie pisemnym zawarto przegląd literaturowy dwóch prac naukowych, które skupiały się na stworzeniu modelu DLS pozwalającego przeprowadzenie klasyfikacji binarnej zdjęć RTG klatki piersiowej dzieci oraz zbadaniu nowego podejścia do modułowej sieci neuronowej, która umożliwiałaby specjalistyczną analizę w modułach oraz zastosowanie jej do problemu cyfrowej analizy obrazu. Przedstawiono materiały i metody wykorzystane w projekcie. Zaprezentowano wykorzystywane biblioteki oraz zbiór danych. Opisano jak przygotowano zbiór danych do dalszej analizy oraz wykorzystane algorytmy. Przedstawiono przebieg prowadzonej klasyfikacji. Posługując się zdjęciami opracowanego kodu zaprezentowano między innymi analizę liczebności zbioru danych, wizualizacje przykładowych zdjęć ze zbioru treningowego oraz wygenerowanie poszczególnych zbiorów. Zaprezentowano implementację sieci własnej oraz trenowanie i ewaluacje modelu. Przedstawiono macierz pomyłek modelu oraz jego wykres ROC. Dokonano implementacji modelu na różnych bazach, do których należały: DenseNet121, VGG16, ResNet50, InceptionV3 i EfficientNetB1. Dla każdej z wymienionych baz przeprowadzono trenowanie modelu oraz jego ewaluację, a następnie wyznaczono macierzy pomyłek i wykres ROC modelu. Przeprowadzono ewaluację otrzymanych wyników oraz zestawiono otrzymane metryki dla wszystkich typów sieci. Przeprowadzono dyskusję otrzymanych wyników oraz podsumowano projekt.

2. Wstęp teoretyczny.

2.1. Spis skrótów.

RTG - zdjęcie rentgenowskie, rentgenogram.
CNN (Convolutional Neural Network) - konwolucyjna sieć neurownowa.
DLS (deep learning system) - system głębokiego uczenia.
TP (true positive) – liczba poprawnie sklasyfikowanych przykładów.
FN (false negative) - decyzja negatywna, podczas gdy przykład w rzeczywistości jest pozytywny.
TN (true negative) – liczba przykładów poprawnie odrzuconych.
FP (false positive) – liczba przykładów błędnie przydzielonych do wybranej klasy,podczas gdy w rzeczywistości do niej nie należą.

2.2. Zapalenie płuc.

Zapalenie płuc to stan zapalny pęcherzyków płucnych lub tkanki podścieliskowej, który charakteryzuje się powstaniem wysięku zapalnego. Skutkuje to zmniejszeniem się powierzchni płuc, przyspieszeniem oddechu, dusznością. Do najczęstszych objawów należą kaszel, ból w klatce piersiowej i świszczący oddech Choroba ta może być powodowana m.in. przez wirusy, bakterie, grzyby, czy też alergię. Wyróżnia się zapalenie płuc:

- odoskrzelowe-drobnoustroje przenikają do organizmu od strony oskrzeli, poprzedza je zwykle zapalenie oskrzeli,
- płatowe-stan zapalny obejmuje jednocześnie jeden płat płuca, a także opłucną, która pokrywa dany płat,
- segmentalne-stan zapalny obejmuje konkretne segmenty płuc [1]

Grupą wiekową, która jest najbardziej narażona na zachorowanie są dzieci, ze względu na niską wydolność układu odpornościowego. U dzieci, zapalenie płuc jest najczęściej wywoływane przez wirusy oraz bakterie, a towarzyszące objawy są niespecyficzne i mogą występować również w innych zakażeniach układu oddechowego. Rozpoznanie choroby opiera się głównie na wykonaniu badania RTG klatki piersiowej. Zmianą potwierdzającą zapalenie płuc są widoczne na zdjęciu rentgenowskim zagęszczenia miąższowe.

Zapalenie płuc jest największą zakaźną przyczyną śmierci dzieci na całym świecie. Dane WHO z 2019 roku pokazują, że choroba ta była przyczyną śmierci 740 180 dzieci w wieku poniżej 5 lat, co stanowiło 14% wszystkich zgonów w tej grupie wiekowej, natomiast wśród dzieci w wieku od 1 do 5 lat było to 22%. Te wysokie statystyki obrazują jak ważna jest prawidłowa diagnoza tej choroby i zastosowanie odpowiedniego leczenia. Zaniedbane lub nie do końca wyleczone zapalenie płuc u dziecka może prowadzić do poważnych powikłań takich jak: wysiękowe zapalenie opłucnej, ropniak opłucnej, ropień płuca, a nawet martwicze zapalenie płuc. [2], [3]

2.3. Sieci konwolucyjne.

Sieć konwolucyjna (CNN) jest klasą sztucznych sieci neuronowych, najczęściej stosowaną do analizy obrazów wizualnych. CNN to uregulowane wersje perceptronów wielowarstwowych, które oznaczają w pełni połączone sieci, czyli każdy neuron w jednej warstwie jest połączony ze wszystkimi neuronami w następnej warstwie. Sieci splotowe zostały zainspirowane procesami biologicznymi, ponieważ wzorzec łączności między neuronami przypomina organizację kory wzrokowej zwierzęcia. Sieci CNN wykorzystują stosunkowo niewiele przetwarzania wstępnego w porównaniu z innymi algorytmami klasyfikacji obrazów. Konwolucyjne sieci neuronowe to wyspecjalizowany rodzaj sztucznych sieci neuronowych, w których w co najmniej jednej warstwie stosuje się operację matematyczną zwaną splotem zamiast ogólnego mnożenia macierzy. Przetwarzają piksele i rozpoznają i przetwarzają obraz. Sieć taka składa się z warstwy wejściowej, warstw ukrytych i warstwy wyjściowej. Głębokie sieci konwolucyjne (CNN) potrafią stopniowo filtrować różne części danych uczących i wyostrzać ważne cechy w procesie dyskryminacji wykorzystanym do rozpoznawania lub klasyfikacji wzorców. W każdej konwolucji można wyróżnić ilość parametrów w każdej warstwie, która zawiera

ilość kanałów, ilość filtrów, szerokość filtra, wysokość filtra, oraz ilość jednostek ukrytych w każdej warstwie, która zawiera ilość filtrów szerokość wzorca, wysokość wzorca. [11]

3. Przegląd literaturowy.

3.1. 'Limited generalizability of deep learning algorithm for pediatric pneumonia classification on external data.'

Celem projektu opisanego w artykule było stworzenie modelu DLS (deep learning system), za pomocą którego możliwe będzie przeprowadzenie klasyfikacji binarnej zdjęć RTG klatki piersiowej dzieci. W badaniu wykorzystano dwa zestawy danych: wewnętrzny, pochodzący z dostępnej publicznie bazy danych z Guangzhou Women and Children's Medical Center w Guangzhou w Chinach zestaw radiogramów pacjentów pediatrycznych w wieku od 1 do 5 lat oraz zewnętrzny, pochodzący z bazy danych ChestXray14, zawierający obrazy RTG klatki piersiowych dzieci z tej samej grupy wiekowej co zestaw wewnętrzny. System został opracowany przy użyciu głębokiej konwolucyjnej sieci neuronowej, korzystając z platformy Keras. Do wyszkolenia głębokiej sieci neuronowej wykorzystano łącznie 5232 radiogramy Po przeprowadzeniu analizy działania nowego modelu DLS, zaobserwowano wysoką skuteczność diagnostyczną w identyfikacji choroby na obrazach medycznych w zestawach testów wewnętrznych, natomiast podczas testowania na zestawie zewnętrznym wydajność modelu była znacznie mniejsza oraz występowały różnice w znaczeniu klinicznym cech wyróżnionych przez mapy cieplne. Sieć osiągnęła wartość AUC 0,95 i 0,54 odpowiednio dla zbioru wewnętrznego i zewnętrznego. Niniejsze badanie podkreśla potencjalne ograniczenia w generalizowaniu takich modeli DLS. [4]

3.2. 'A new modular neural network approach with fuzzy response integration for lung disease classification based on multiple objective feature optimization in chest X-ray images.'

Głównym założeniem tego badania jest nowe podejście modułowej sieci neuronowej, wykorzystujące integrację odpowiedzi rozmytej opartej na podziale cech w celu uzyskania specjalistycznej analizy w modułach, oraz jej zastosowanie do problemu cyfrowej analizy obrazu. Ponadto, korzystając z wielokryterialnego algorytmu genetycznego, można znaleźć bardziej odpowiednią grupę cech spośród 84 dostępnych wstępnie wybranych cech uzyskanych metodami ekstrakcji cech. Umożliwia również znalezienie wektorów rozwiązań dla zadania klasyfikacji zapalenia płuc i klasyfikacji guzków płucnych.

Jest to alternatywa dla problemu wieloklasowego o niskiej dokładności, gdzie trenuje się serię zoptymalizowanych binarnie klasyfikatorów, z których każdy specjalizuje się w jednej chorobie. Klasyfikatory te działają jako modułowa sieć neuronowa. Dla zapalenia płuc oraz choroby guzków płuc zastosowano klasyfikator binarny, dodatkowo jest możliwość poszerzenia całości badań i stworzenie klasyfikatora binarnego, który będzie wykrywał każdą inną chorobę na podstawie prześwietlenia klatki piersiowej. Takie podejście ma rozwiązać problem klasyfikacji wieloklasowej za pomocą serii połączonych i wyspecjalizowanych modeli sieci neuronowych. Specjalistyczne moduły uzyskują informacje RTG klatki piersiowej poprzez ekstrakcję i selekcję cechy obrazu w skali szarości, aby uzyskać potencjalną grupę cech.

W radiografii klatki piersiowej na zapalenie płuc mogą wskazywać białe zmętnienia lub chmury istoty białej w przestrzeni powietrznej płuc, pojawiają się one ponieważ

promieniowanie nie może przejść przez gęste części ciała, takie jak kości, co powoduje, że miejsce z zapaleniem płuc wygląda na białe. Normalny obszar powietrza płuc jest zawsze czarnym obrazem na RTG. Aby ocenić poprawność metody przetestowano klasyfikator na dwóch zestawach danych pierwszy to ChestXRay14 autorstwa Wanga, a drugi ZhangLab autorstwa Kermany, które następnie podzielono na dwie klasy: jedna to "zapalenie płuc", natomiast druga to "normalne płuca". Podzbiory podzielono w stosunku 70:30 dla treningu i testowania. Celem było wyszkolenie klasyfikatora, aby nauczył się wykrywać chorobę w oparciu o różnice między klasą z objawami zapalenia a normalną klasą.

Guzki płucne to małe okrągłe białe plamki od 2 do 3 cm, które pojawiają się w przestrzeni powietrznej płuc na zdjęciu rentgenowskim. Ponad 90% wszystkich wykrytych guzków płucnych o średnicy mniejszej niż 3 cm są łagodne. Aby przetestować metodę wyszkolono klasyfikator na zbiorze danych JSRT autorstwa Shiraishi ze współautorami, który jest często używany do diagnozowania guzków płuc na RTG. Zbiór 247 obrazów podzielono na dwie klasy, tą 154 obrazami "guzki płucne" i 93 obrazami "normalnymi".

Przedstawiony w artykule nowy zoptymalizowany hybrydowy model neuro-rozmyty ma zdolność do osiągnięcia od minimum 95% dokładności klasyfikacji aż do 99,83% dla klasyfikacji obrazów medycznych do wykrywania chorób płuc. Dany klasyfikator przewyższył inne metody pod względem dokładności klasyfikacji wykorzystując cechy obrazu jako główne źródło dyskryminacji klasowej. Dobrze sprawdza się w rozdzielaniu zdrowych i niezdrowych zdjęć rentgenowskich klatki piersiowej poprzez znajdowanie różnic w intensywności pikseli. Model hybrydowy, łączy modułową sieć neuronową z systemem rozmytym w celu integracji wyjść modułów, a wieloobiektowy algorytm genetyczny używany jest do wyboru najistotniejszych cech, które mają być wykorzystane jako dane wejściowe dla modularnej sieci neuronowej. [10]

4. Materiały i metody.

Projekt został zrealizowany w języku programistycznym Python 3. Dzięki swojej wysokiej czytelności oraz przystępności odznacza on się dużą popularnością. Dlatego też posiada bardzo szeroką jak i zróżnicowaną liczbę bibliotek, dającą ogromny wachlarz możliwości.

4.1. Wykorzystane biblioteki.

Poniżej przedstawiono popularne biblioteki, którymi posłużono się podczas realizacji projektu. Należą do nich:

- **NumPy** - biblioteka, pozwalająca wykonywać szybkie operacje na macierzach, w tym matematyczne, logiczne, sortujące, szybką transformatę Fouriera
- **MatPlotLib** - biblioteka oparta na bibliotece NumPy, pomocna przy tworzeniu statystycznych, animowanych i interaktywnych wizualizacji w Pythonie
- **Sklearn** - biblioteka zbudowana m.in. na NumPy, MatPlotLib, jest prostym i wydajnym narzędziem do predykcyjnej analizy danych, wykorzystywana w uczeniu maszynowym

- **TensorFlow** - biblioteka umożliwiająca tworzenie modeli uczenia maszynowego
- **Keras** - biblioteka zbudowana na bazie TensorFlow, używana do głębokiego uczenia w Pythonie, jego zaletą jest możliwość szybkiego eksperymentowania, zmniejszenia obciążenia poznawczego programistów, wysoka wydajność [5]–[9]

4.2. Zbiór danych.

Dane wykorzystane w projekcie stanowił zbiór uzyskany ze strony Kaggle [Zbiór danych]. Zbiór ten przedstawiał rutynowe zdjęcia RTG klatki piersiowej dzieci w wieku 5 lat ze szpitala Guangzhou w Chinach. Zawierał on 5856 zdjęć i został podzielony na trzy zbiory: treningowy, testowy oraz walidacyjny. Zbiór treningowy zawierał 3110 zdjęć z rozpoznanym zapaleniem płuc oraz 1082 zdjęć z obrazem zdrowych płuc. Zbiór testowy posiadał 390 zdjęć z rozpoznanym zapaleniem płuc oraz 234 zdjęć z obrazem zdrowych płuc. Zbiór walidacyjny posiadał 773 zdjęcia z rozpoznanym zapaleniem płuc oraz 267 zdjęć z obrazem zdrowych płuc. Dokonano binaryzacji obrazów. Przed przystąpieniem do dalszej części projektu zmieniono rozmiar zdjęć do (180,180,3).

Zbiór danych został odpowiednio przygotowany. Przed rozpoczęciem treningu przystąpiono do modyfikacji obrazów w taki sposób, aby lepiej nadawały się treningu konwencjonalnej sieci neuronowej. W tym celu użyto funkcji z biblioteki TensorFlow *ImageDataGenerator*, która służy do przetwarzania i powiększania danych.

Klasa ta dodatkowo zapewnia wsparcie dla podstawowego rozszerzenia danych, takiego jak losowe odwracanie obrazów w poziomie. Za pomocą tego generatora możliwe jest również przekształcenie wartości w każdej partii tak, aby ich średnia wynosiła 0, a odchylenie standardowe 1, co ułatwia trening modelu poprzez standaryzację rozkładu wejściowego. Generator ten umożliwia również przekształcenie jednokanałowych obrazów rentgenowskich (w skali szarości) na format trzykanałowy, poprzez powtarzanie wartości na obrazach we wszystkich kanałach. Jest to konieczny zabieg, ponieważ wstępnie wytrenowany model, który będzie później wykorzystywany wymaga trójkanałowych danych wejściowych.

Zarówno dla zbioru walidacyjnego jak i dla zbioru testowego został zbudowany osobny generator. Stało się tak, ponieważ generator który został wykorzystany dla danych treningowych normalizuje każdy obraz na partię, co oznacza, że wykorzystuje on statystyki partii. Taki zabieg jest niemożliwy do wykorzystywania dla danych testowych i danych walidacyjnych, ponieważ w prawdziwym życiu nie przetwarzamy obrazów przychodzących po kolei, tylko pojedynczo. Znajomość wartości średniej na partie danych testowych dałaby wykorzystywanemu modelowi przewagę (model nie powinien posiadać żadnych informacji odnośnie danych testowych). W związku z tym należy znormalizować dane testowe, wykorzystując przy tym statystyki obliczone na podstawie zbioru treningowego.

4.3. Algorytmy.

Dokonano implementacji własnej sieci konwolucyjnej, składającej się 24 warstw. Spośród nich możemy wyróżnić:

- tf.keras.layers.Conv2D() - warstwa tworzy jądro splotu, które jest połączone z wejściem warstwy w celu wytworzenia tensora wyjść
- tf.keras.layers.BatchNormalization() - warstwa, która normalizuje swoje dane wejściowe, stosuje transformację, która utrzymuje średnią wartość wyjścia bliską 0, a odchylenie standardowe wyjścia bliskie 1
- tf.keras.layers.MaxPool2D() - warstwa próbkuje dane wejściowe wzdłuż ich wymiarów przestrzennych, przyjmując maksymalną wartość w oknie wejściowym
- tf.keras.layers.Flatten() - warstwa spłaszczająca dane wejściowe
- tf.keras.layers.Dense() - warstwa, która jest głęboko połączona, oznacza to, że każdy neuron w warstwie otrzymuje dane wejściowe wszystkich neuronów poprzedniej warstwy
- tf.keras.layers.Dropout() - warstwa ustawia jednostki wejściowe na 0 z określoną częstotliwością, na każdym kroku treningu, co pomaga zapobiec nadmiernemu dopasowaniu

Korzystano z dostępnych modeli przetrenowanych już sieci. Pierwszą z nich była sieć DenseNet121, której architektura opiera się na tym, ze w każdej warstwie mapy obiektów ze wszystkich poprzednich warstw nie są sumowane, ale łączone i używane jako dane wejściowe. W konsekwencji sieć DenseNet121 wymaga mniejszej liczby parametrów niż tradycyjne CNN. Posiada ona 124 warstwy: 120 warstw Convolutional i 4 warstwy AvgPool. Zaletą tej sieci jest fakt, że wymaga mniejszej liczby parametrów i umożliwia ponowne wykorzystanie funkcji, zapewniając bardziej kompaktowe modele i osiągając lepsze wyniki w konkurencyjnych zestawach danych. https://iq.opengenus.org/architecture-of-densenet121/

Kolejnym modelem była sieć VGG16, charakteryzuje się ona bardzo jednolitą architekturą, składa się z 16 warstw splotowych. Jest najbardziej preferowaną siecią w dziedzinie ekstrahowania cech. Jedyną jej wadą jest duża ilość parametrów - 138, co może powodować trudności w obsłudze sieci. Została wprowadzona w 2014 roku. https://iq.opengenus.org/vgg16/

Sieć ResNet50 jest bardzo dobrze zbadanym i szeroko używanym modelem, użytym pierwszy raz w 2012 roku. Sprawdza się w klasyfikacji obrazów, lokalizacji obiektów, wykrywaniu obiektów. Składa się z 50 warstw Convolutional. https://iq.opengenus.org/resnet50-architecture/

Model sieci InceptionV3 powstał jako moduł dla GoogleNet, dzięki niej jest możliwa analiza obrazów i wykrywanie obiektów. Jest już trzecią z kolei modyfikacją podstawowego modułu InceptionV1, został wydany w 2015 roku, zawiera 42 warstwy, charakteryzuje się wysoką wydajnością i mniejszymi kosztami obliczeniowymi, wykazuje również niski poziom błędów. https://iq.opengenus.org/inception-v3-model-architecture/

EfficientNetB1 należy do rodziny sieci EfficientNet, została wprowadzona w 2019 roku, więc jest najnowszą z użytych sieci, i jedną z najbardziej wydajnych. Została opracowana na modelu bazowym EfficientNetB0, który został stworzony przez samą sieć neuronową. Sieci z tej rodziny osiągnęły najnowocześniejszą dokładność.

https://datamonje.medium.com/image-classification-with-efficientnet-better-performance-with-computational-efficiency-f480fdb00ac6

Przekształcono wyjście z sieci w taki sposób, aby były one lepiej dopasowane do klasyfikacji zdjęć wybranego zbioru zdjęć. W szczególności zwrócono uwagę na ostatnią warstwę tf.keras.layers.Dense(), która odpowiedzialna była za klasyfikowanie na zbiór zdjęć chorych oraz zbiór zdjęć zdrowych.

5. Przebieg klasyfikacji.

1. Pobieranie bibliotek.

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn.utils
from tensorflow import keras

from keras.preprocessing.image import ImageDataGenerator

from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay

from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Dropout, Flatten,
BatchNormalization, GlobalAveragePooling2D

from keras.applications.densenet import DenseNet121
from keras.applications.vgg16 import VGG16
from keras.applications.resnet import ResNet50
from keras.applications import InceptionV3
from keras.applications import EfficientNetB1
```

2. Analiza liczebności zbiorów danych.

```
#Dostęp do zdjęć
train_path = "/content/drive/My Drive/TOM -
projekt/DANE/better/train"
test_path = "/content/drive/My Drive/TOM - projekt/DANE/better/test"
```

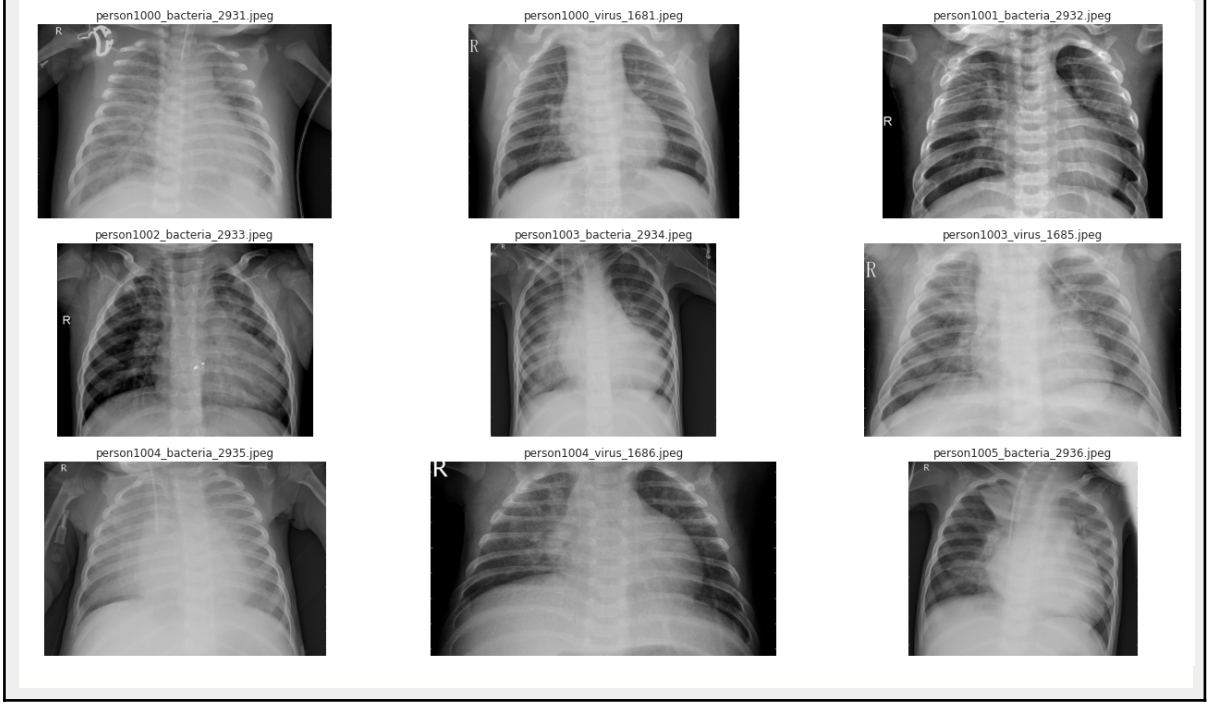
```
val_path = "/content/drive/My Drive/TOM - projekt/DANE/better/val"

#Liczba zdjęć w folderze train
print("Train set:")
print(f"Pneumonia = {len(os.listdir(os.path.join(train_path,
'opacity')))}")
print(f"Normal = {len(os.listdir(os.path.join(train_path,
'normal')))}")

#Liczba zdjęć w folderze test
print("\nTest set:")
print(f"Pneumonia = {len(os.listdir(os.path.join(test_path,
'opacity')))}")
print(f"Normal = {len(os.listdir(os.path.join(test_path,
'normal')))}")

#Liczba zdjęć w folderze val
print("\nValidation set:")
print(f"Pneumonia = {len(os.listdir(os.path.join(val_path,
'opacity')))}")
print(f"Normal = {len(os.listdir(os.path.join(val_path,
'normal')))}")
```

```
Train set:
Pneumonia = 3110
Normal = 1082

Test set:
Pneumonia = 390
Normal = 234

Validation set:
Pneumonia = 773
Normal = 267
```

3. Wizualizacja przykładowych 9 zdjęć ze zbioru “train” - zapalenie płuc.

```
pneumonia_path = "/content/drive/My Drive/TOM -
projekt/DANE/better/train/opacity"
pneumonia_img = os.listdir("/content/drive/My Drive/TOM -
projekt/DANE/better/train/opacity")
```

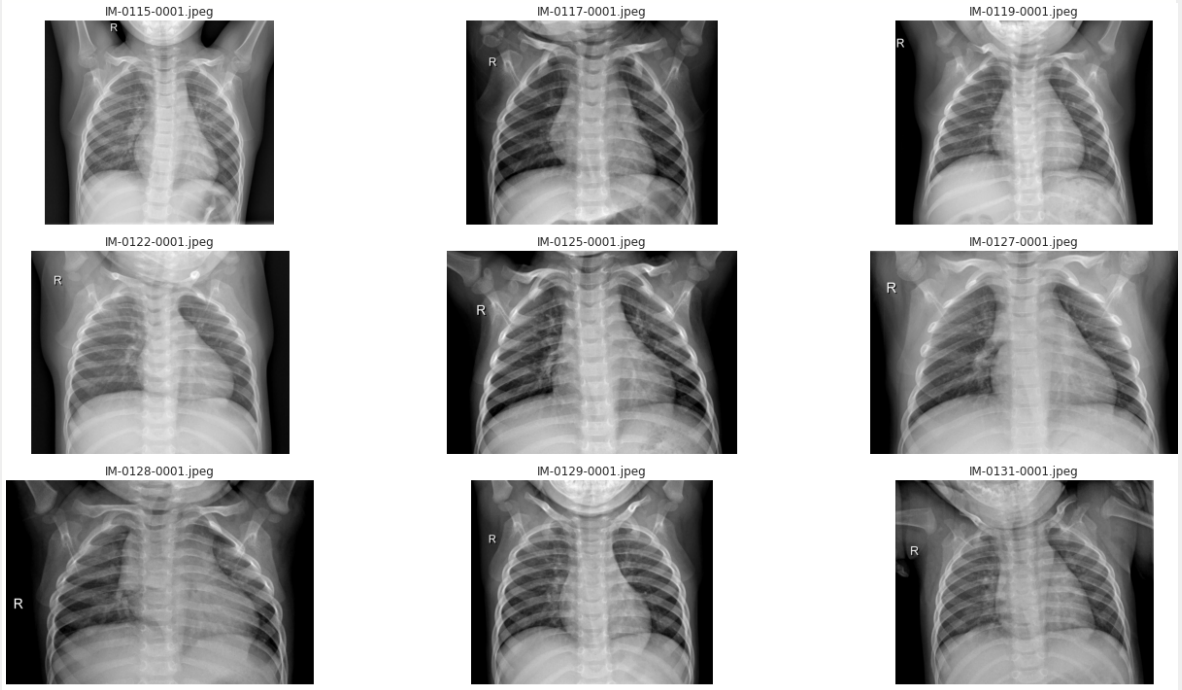
```
plt.figure(figsize=(20, 10))
for i in range(9):
    plt.subplot(3, 3, i+1)
    img = plt.imread(os.path.join(pneumonia_path, pneumonia_img[i]))
    plt.title(pneumonia_img[i])
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
```



4. Wizualizacja przykładowych 9 zdjęć ze zbioru “train” - płuca zdrowe.

```
normal_path = "/content/drive/My Drive/TOM -
projekt/DANE/better/train/normal"
normal_img = os.listdir("/content/drive/My Drive/TOM -
projekt/DANE/better/train/normal")

plt.figure(figsize=(20, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(normal_path, normal_img[i]))
    plt.title(normal_img[i])
    plt.imshow(img, cmap='gray')
    plt.axis('off')
```

```
plt.tight_layout()
```



5. Analiza pikseli pojedynczego zdjęcia.

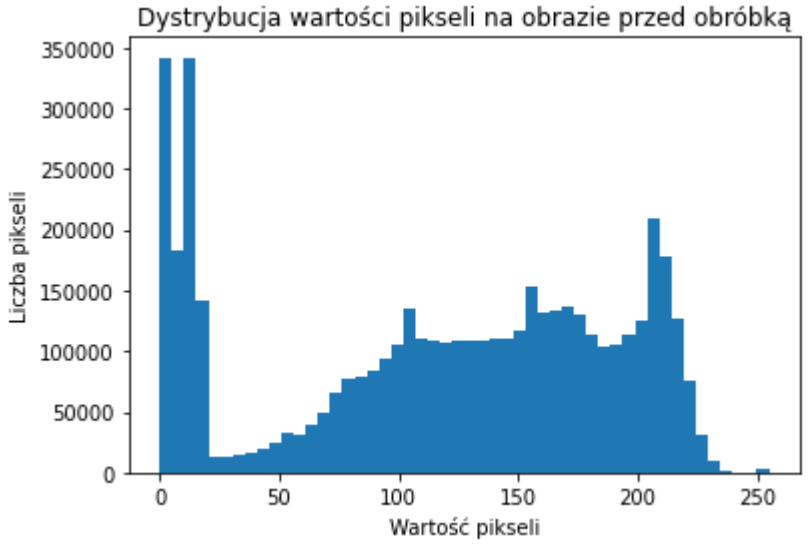
```
example_path = "/content/drive/My Drive/TOM -
projekt/DANE/better/train/normal"
example_img = os.listdir("/content/drive/My Drive/TOM -
projekt/DANE/better/train/normal")

sample_img = plt.imread(os.path.join(example_path, example_img[1]))

plt.imshow(sample_img, cmap='gray')
plt.colorbar()
plt.title('Obraz przed obróbką')

print(f"Rozmiar zdjęcia: {sample_img.shape}")
print(f"Max pixel: {sample_img.max():.2f}")
print(f"Min pixel {sample_img.min():.2f}")
print(f"Średnia: {sample_img.mean():.2f} ")
print(f"Odchylenie standardowe: {sample_img.std():.2f}")
```

```
Rozmiar zdjęcia: (1152, 1422)
Max pixel: 255.00
Min pixel 0.00
Średnia: 100.65
```

```
Odchylenie standardowe: 59.81
```

6. Histogram wartości pikseli.

```
plt.hist(sample_img.ravel(), bins = 50)

plt.title('Dystrybucja wartości pikseli na obrazie przed obróbką')
plt.xlabel('Wartość pikseli')
plt.ylabel('Liczba pikseli')
plt.show()
```



7. Obróbka zdjęć, wygenerowanie zbiorów: treningowego, testowego i walidacyjnego.

```
img_gen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    samplewise_center=True,
    samplewise_std_normalization=True)

train = img_gen.flow_from_directory(
    train_path,
    batch_size=8,
    shuffle=True,
    class_mode='binary',
    target_size=(180, 180))

validation = img_gen.flow_from_directory(
    val_path,
    batch_size=1,
    shuffle=False,
    class_mode='binary',
    target_size=(180, 180))

test = image_generator.flow_from_directory(
    test_path,
    batch_size=1,
    shuffle=False,
    class_mode='binary',
    target_size=(180, 180))
```

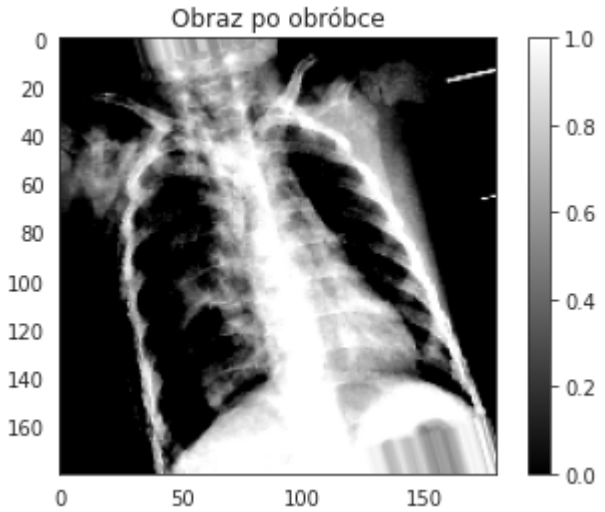
```
Found 4192 images belonging to 2 classes.
Found 1040 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

8. Analiza pikseli pojedynczego zdjęcia po transformacji.

```
trans_img, label = train.__getitem__(0)

plt.imshow(trans_img[1], cmap='gray')
plt.colorbar()
plt.title('Obraz po obróbce')
print(f"Rozmiar zdjęcia: {trans_img.shape}")
```

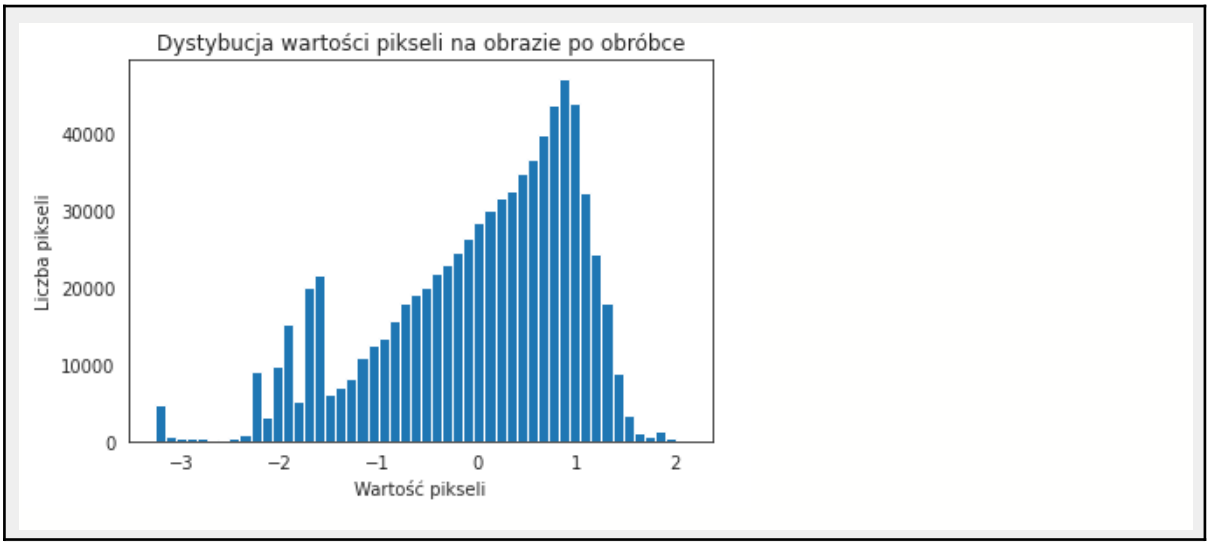
```
print(f"Max pixel: {trans_img.max():.2f}")
print(f"Min pixel {trans_img.min():.2f}")
print(f"Średnia: {trans_img.mean():.2f} ")
print(f"Odchylenie standardowe: {trans_img.std():.2f}")
```

```
Rozmiar zdjęcia: (8, 180, 180, 3)
Max pixel: 2.36
Min pixel -5.88
Średnia: 0.00
Odchylenie standardowe: 1.00
```



9. Histogram wartości pikseli po transformacji zdjęcia.

```
plt.hist(trans_img.ravel(), bins = 50)

plt.title('Dystrybucja wartości pikseli na obrazie po obróbce')
plt.xlabel('Wartość pikseli')
plt.ylabel('Liczba pikseli')
plt.show()
```



10. Wyznaczenie wag dla klas w zbiorze treningowym.

```
class_weights = sklearn.utils.class_weight.compute_class_weight(
            'balanced',
             classes = np.unique(train.classes),
             y = train.classes)

class_weights = dict(zip(np.unique(train.classes), class_weights))
print(class_weights)
```

```
{0: 1.9371534195933457, 1: 0.6739549839228296}
```

11. Przygotowanie metryk oraz optymalizatora.

```
METRICS = ['accuracy',
           tf.keras.metrics.Precision(name='precision'),
           tf.keras.metrics.Recall(name='recall'),
           tf.keras.metrics.AUC(name='AUC')]

adam_001 = tf.keras.optimizers.Adam(learning_rate=0.001)
```

12. Implementacja sieci własnej.

```
our_model = Sequential()

our_model.add(Conv2D(filters=16, kernel_size=(3, 3),
input_shape=(180, 180, 3), activation='relu'))
our_model.add(BatchNormalization())
our_model.add(Conv2D(filters=16, kernel_size=(3, 3),
input_shape=(180, 180, 3), activation='relu'))
our_model.add(BatchNormalization())
our_model.add(MaxPool2D(pool_size=(2, 2)))

our_model.add(Conv2D(filters=32, kernel_size=(3, 3),
activation='relu'))
our_model.add(BatchNormalization())
our_model.add(Conv2D(filters=32, kernel_size=(3, 3),
activation='relu'))
our_model.add(BatchNormalization())
our_model.add(MaxPool2D(pool_size=(2, 2)))

our_model.add(Conv2D(filters=64, kernel_size=(3, 3),
activation='relu'))
our_model.add(BatchNormalization())
our_model.add(Conv2D(filters=64, kernel_size=(3, 3),
activation='relu'))
our_model.add(BatchNormalization())
our_model.add(MaxPool2D(pool_size=(2, 2)))

our_model.add(Conv2D(filters=128, kernel_size=(3, 3),
activation='relu'))
our_model.add(BatchNormalization())
our_model.add(Conv2D(filters=128, kernel_size=(3, 3),
activation='relu'))
our_model.add(BatchNormalization())
our_model.add(MaxPool2D(pool_size=(2, 2)))

our_model.add(Flatten())
our_model.add(Dense(128, activation='relu'))
our_model.add(Dropout(0.3))

our_model.add(Dense(1, activation='sigmoid'))
```

```
our_model.compile(loss = 'binary_crossentropy',
                  optimizer = 'adam',
                  metrics = METRICS)
```

13. Trenowanie modelu.

```
our_r = our_model.fit(
    train,
    epochs = 10,
    validation_data = validation,
    class_weight = class_weights,
    steps_per_epoch = 100,
    validation_steps = 25)
```

```
Epoch 1/10
100/100 [==============================] - 19s 176ms/step - loss:
0.6969 - accuracy: 0.8300 - precision: 0.9415 - recall: 0.8320 - AUC:
0.8821
Epoch 2/10
100/100 [==============================] - 17s 170ms/step - loss:
0.2735 - accuracy: 0.8913 - precision: 0.9630 - recall: 0.8859 - AUC:
0.9576
Epoch 3/10
100/100 [==============================] - 17s 171ms/step - loss:
0.3512 - accuracy: 0.8687 - precision: 0.9495 - recall: 0.8609 - AUC:
0.9411
Epoch 4/10
100/100 [==============================] - 16s 163ms/step - loss:
0.3154 - accuracy: 0.8725 - precision: 0.9540 - recall: 0.8708 - AUC:
0.9463
Epoch 5/10
100/100 [==============================] - 17s 173ms/step - loss:
0.3228 - accuracy: 0.8950 - precision: 0.9675 - recall: 0.8904 - AUC:
0.9541
Epoch 6/10
100/100 [==============================] - 16s 162ms/step - loss:
0.3426 - accuracy: 0.8875 - precision: 0.9632 - recall: 0.8822 - AUC:
0.9447
Epoch 7/10
100/100 [==============================] - 20s 201ms/step - loss:
0.2458 - accuracy: 0.8975 - precision: 0.9688 - recall: 0.8902 - AUC:
```

```
0.9646
Epoch 8/10
100/100 [==============================] - 18s 173ms/step - loss:
0.3848 - accuracy: 0.8512 - precision: 0.9520 - recall: 0.8407 - AUC:
0.9317
Epoch 9/10
100/100 [==============================] - 19s 193ms/step - loss:
0.1972 - accuracy: 0.9200 - precision: 0.9856 - recall: 0.9070 - AUC:
0.9726
Epoch 10/10
100/100 [==============================] - 16s 161ms/step - loss:
0.1549 - accuracy: 0.9400 - precision: 0.9825 - recall: 0.9365 - AUC:
0.9854
```

14. Ewaluacja modelu.

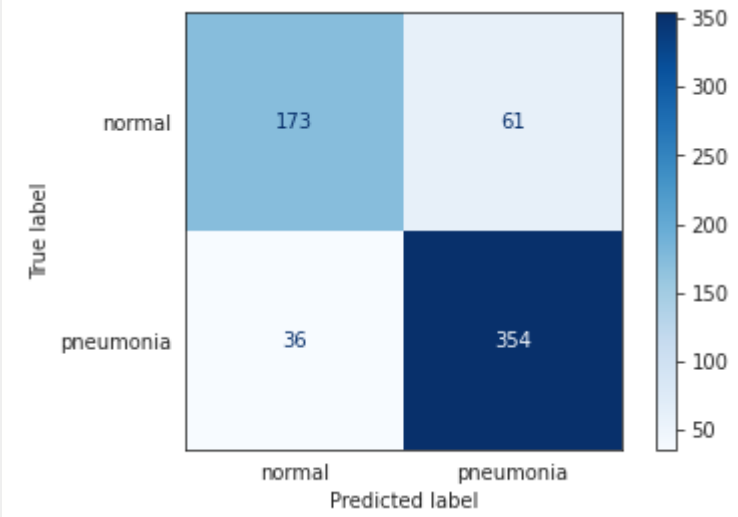
```
our_evaluation = our_model.evaluate(test)

print(f"Test Accuracy: {our_evaluation[1]:.2f}")
print(f"Test Precision: {our_evaluation[2]:.2f}")
print(f"Test Recall: {our_evaluation[3]:.2f}")
print(f"Test AUC: {our_evaluation[4]:.2f}")
```

```
624/624 [==============================] - 13s 21ms/step - loss:
0.8065 - accuracy: 0.8526 - precision: 0.8548 - recall: 0.9205 - AUC:
0.8948
Test Accuracy: 0.85%
Test Precision: 0.85%
Test Recall: 0.92%
Test AUC: 0.89%
```

15. Macierz pomyłek modelu.

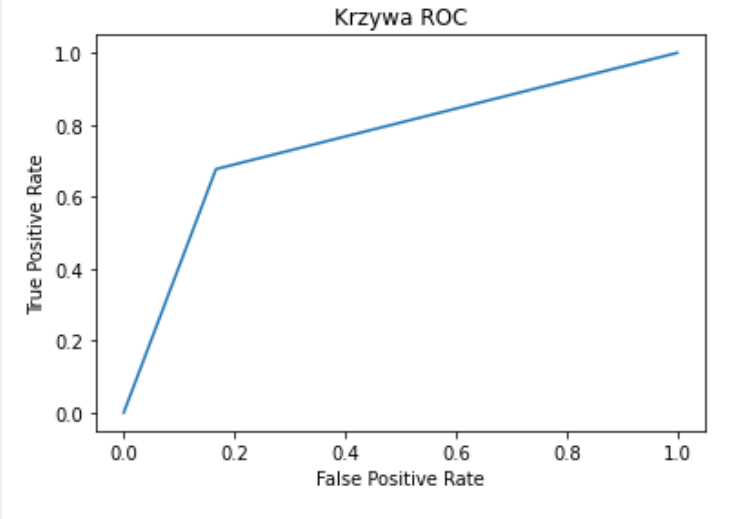
```
our_pred = our_model.predict(test)

matrix = confusion_matrix(test.classes, (our_pred > 0.5).ravel())

ConfusionMatrixDisplay(matrix, display_labels = ["normal",
"pneumonia"]).plot(cmap = plt.cm.Blues)
```

```
plt.show()
```



16. Wykres ROC modelu.

```
fpr, tpr, _ = metrics.roc_curve(test.classes, (our_pred >
0.5).ravel())

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



17. Implementacja modelu na bazie DenseNet121.

```
dense121_base_model = DenseNet121(input_shape=(180, 180, 3),
include_top=False, weights='imagenet')

x = dense121_base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(1, activation="sigmoid")(x)
model = Model(inputs=dense121_base_model.input, outputs=predictions)

dense_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=METRICS)
```

18. Trenowanie modelu.

```
dense_r = dense_model.fit(
    train,
    epochs=10,
    validation_data=validation,
    class_weight=class_weights,
    steps_per_epoch=100,
    validation_steps=25)
```

```
Epoch 1/10
100/100 [==============================] - 44s 297ms/step - loss:
0.7231 - accuracy: 0.5738 - precision: 0.8205 - recall: 0.5415 - AUC:
0.6579
Epoch 2/10
100/100 [==============================] - 32s 317ms/step - loss:
0.4980 - accuracy: 0.7200 - precision: 0.9454 - recall: 0.6644 - AUC:
0.8623
Epoch 3/10
100/100 [==============================] - 25s 249ms/step - loss:
0.3973 - accuracy: 0.8075 - precision: 0.9476 - recall: 0.7860 - AUC:
```

```
0.9075
Epoch 4/10
100/100 [==============================] - 25s 252ms/step - loss:
0.3760 - accuracy: 0.8375 - precision: 0.9538 - recall: 0.8236 - AUC:
0.9158
Epoch 5/10
100/100 [==============================] - 23s 226ms/step - loss:
0.3955 - accuracy: 0.8388 - precision: 0.9421 - recall: 0.8313 - AUC:
0.9090
Epoch 6/10
100/100 [==============================] - 24s 244ms/step - loss:
0.4123 - accuracy: 0.8462 - precision: 0.9485 - recall: 0.8381 - AUC:
0.8997
Epoch 7/10
100/100 [==============================] - 22s 221ms/step - loss:
0.3946 - accuracy: 0.8250 - precision: 0.9381 - recall: 0.8179 - AUC:
0.9090
Epoch 8/10
100/100 [==============================] - 22s 220ms/step - loss:
0.3573 - accuracy: 0.8225 - precision: 0.9520 - recall: 0.8091 - AUC:
0.9196
Epoch 9/10
100/100 [==============================] - 23s 230ms/step - loss:
0.3153 - accuracy: 0.8700 - precision: 0.9642 - recall: 0.8576 - AUC:
0.9372
Epoch 10/10
100/100 [==============================] - 21s 209ms/step - loss:
0.3156 - accuracy: 0.8587 - precision: 0.9768 - recall: 0.8336 - AUC:
0.9357
```

19. Ewaluacja modelu.

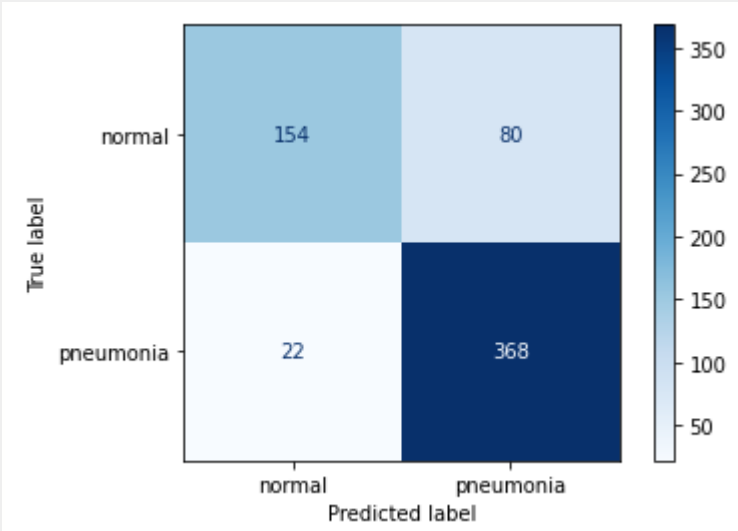
```
dense_evaluation= dense_model.evaluate(test)

print(f"Test Accuracy: {dense_evaluation[1]:.2f}")
print(f"Test Precision: {dense_evaluation[2]:.2f}")
print(f"Test Recall: {dense_evaluation[3]:.2f}")
print(f"Test AUC: {dense_evaluation[4]:.2f}")
```

```
624/624 [==============================] - 20s 32ms/step - loss:
0.3975 - accuracy: 0.8429 - precision: 0.8274 - recall: 0.9462 - AUC:
```

```
0.9169
Test Accuracy: 0.84
Test Precision: 0.83
Test Recall: 0.95
Test AUC: 0.92
```

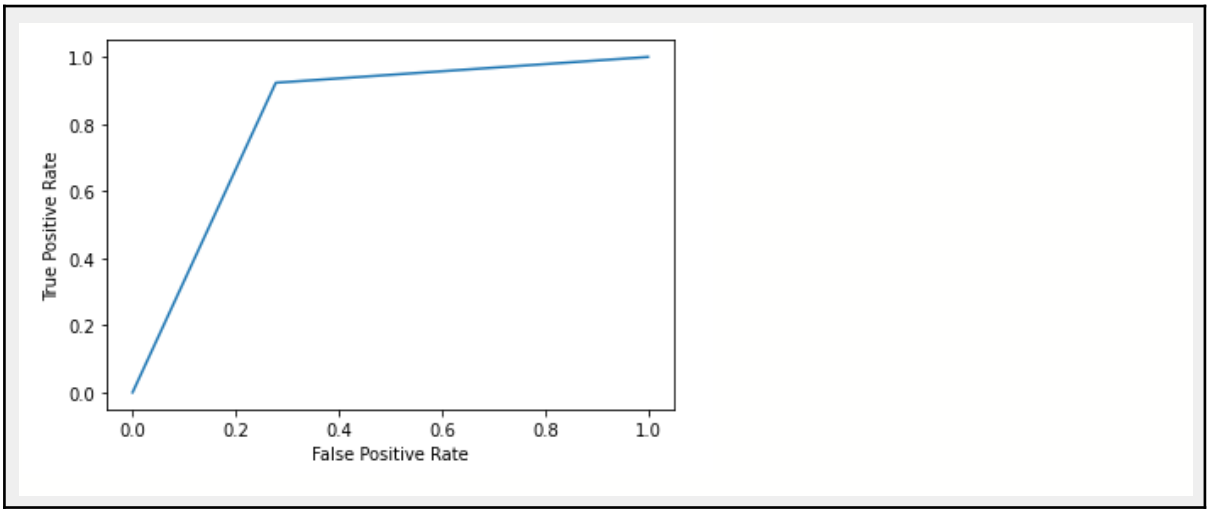
20. Macierz pomyłek modelu.

```
dense_pred = dense_model.predict(test)

matrix = confusion_matrix(test.classes, (dense_pred > 0.7).ravel())

ConfusionMatrixDisplay(matrix, display_labels = ["normal",
"pneumonia"]).plot(cmap = plt.cm.Blues)
plt.show()
```



21. Wykres ROC modelu.

```
fpr, tpr, _ = metrics.roc_curve(test.classes, (dense_pred >
0.5).ravel())

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



22. Implementacja modelu na bazie VGG16.

```
vgg16_base_model = VGG16(input_shape=(180,180,3), include_top=False,
weights='imagenet')

vgg_model = tf.keras.Sequential([
        vgg16_base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(64, activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(32,activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(1,activation="sigmoid")])

vgg_model.compile(loss='binary_crossentropy',
                  optimizer = adam_001,
                  metrics = METRICS)
```

23. Trenowanie modelu.

```
vgg_r = vgg_model.fit(
    train,
    epochs = 10,
    validation_data = validation,
```

```
    class_weight = class_weights,
    steps_per_epoch = 100,
    validation_steps = 25)
```

```
Epoch 1/10
100/100 [==============================] - 22s 221ms/step - loss:
0.7329 - accuracy: 0.5888 - precision: 0.8321 - recall: 0.5542 - AUC:
0.6779
Epoch 2/10
100/100 [==============================] - 21s 209ms/step - loss:
0.5488 - accuracy: 0.6975 - precision: 0.9381 - recall: 0.6359 - AUC:
0.8162
Epoch 3/10
100/100 [==============================] - 21s 210ms/step - loss:
0.5563 - accuracy: 0.7237 - precision: 0.9152 - recall: 0.6982 - AUC:
0.8072
Epoch 4/10
100/100 [==============================] - 21s 205ms/step - loss:
0.5527 - accuracy: 0.7262 - precision: 0.9185 - recall: 0.6962 - AUC:
0.8004
Epoch 5/10
100/100 [==============================] - 21s 209ms/step - loss:
0.5063 - accuracy: 0.7487 - precision: 0.9316 - recall: 0.7207 - AUC:
0.8283
Epoch 6/10
100/100 [==============================] - 20s 196ms/step - loss:
0.4547 - accuracy: 0.7812 - precision: 0.9519 - recall: 0.7496 - AUC:
0.8651
Epoch 7/10
100/100 [==============================] - 20s 198ms/step - loss:
0.4252 - accuracy: 0.7937 - precision: 0.9388 - recall: 0.7731 - AUC:
0.8787
Epoch 8/10
100/100 [==============================] - 20s 200ms/step - loss:
0.4187 - accuracy: 0.8075 - precision: 0.9637 - recall: 0.7670 - AUC:
0.8861
Epoch 9/10
100/100 [==============================] - 21s 208ms/step - loss:
0.4537 - accuracy: 0.7700 - precision: 0.9376 - recall: 0.7455 - AUC:
0.8654
Epoch 10/10
100/100 [==============================] - 19s 193ms/step - loss:
0.4027 - accuracy: 0.7912 - precision: 0.9529 - recall: 0.7635 - AUC:
```

```
0.8864
```

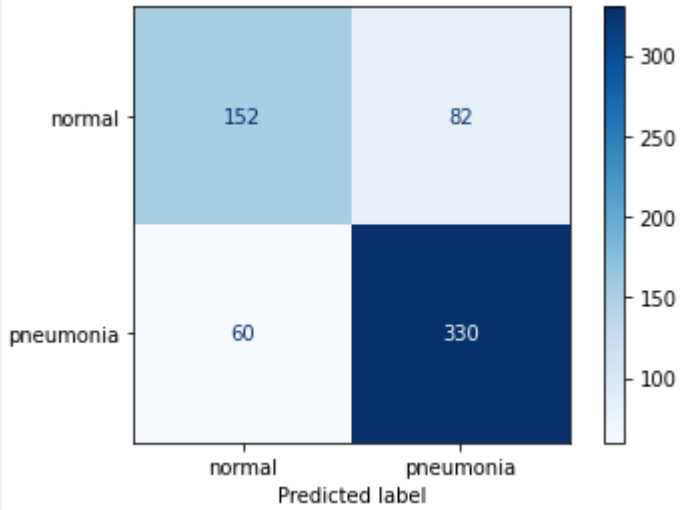
24. Ewaluacja modelu.

```
vgg_evaluation = vgg_model.evaluate(test)

print(f"Test Accuracy: {vgg_evaluation[1]:.2f}")
print(f"Test Precision: {vgg_evaluation[2]:.2f}")
print(f"Test Recall: {vgg_evaluation[3]:.2f}")
print(f"Test AUC: {vgg_evaluation[4]:.2f}")
```

```
624/624 [==============================] - 15s 24ms/step - loss:
0.5798 - accuracy: 0.7596 - precision: 0.7553 - recall: 0.9103 - AUC:
0.8563
Test Accuracy: 0.76
Test Precision: 0.76
Test Recall: 0.91
Test AUC: 0.86
```

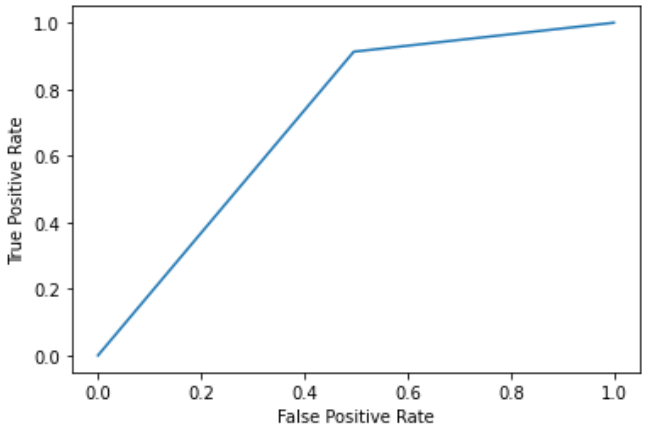
25. Macierz pomyłek modelu.

```
vgg_pred = vgg_model.predict(test)

matrix = confusion_matrix(test.classes, (vgg_pred > 0.7).ravel())

ConfusionMatrixDisplay(matrix, display_labels = ["normal",
"pneumonia"]).plot(cmap = plt.cm.Blues)
plt.show()
```



26. Wykres ROC modelu.

```
fpr, tpr, _ = metrics.roc_curve(test.classes, (vgg_pred >
0.5).ravel())

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



27. Implementacja modelu na bazie ResNet50.

```
resnet50_base_model = ResNet50(input_shape=(180,180,3),
include_top=False, weights='imagenet')

resnet_model = tf.keras.Sequential([
        resnet50_base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(64, activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(32,activation="relu"),
        BatchNormalization(),
```

```
        Dropout(0.2),
        Dense(1,activation="sigmoid")])

resnet_model.compile(loss='binary_crossentropy',
                     optimizer = adam_001,
                     metrics = METRICS)
```

28. Trenowanie modelu.

```
resnet_r = resnet_model.fit(
    train,
    epochs = 10,
    validation_data = validation,
    class_weight = class_weights,
    steps_per_epoch = 100,
    validation_steps = 25)
```

```
Epoch 1/10
100/100 [==============================] - 32s 248ms/step - loss:
0.6791 - accuracy: 0.6413 - precision: 0.8479 - recall: 0.5954 - AUC:
0.7360
Epoch 2/10
100/100 [==============================] - 22s 222ms/step - loss:
0.6800 - accuracy: 0.6488 - precision: 0.8827 - recall: 0.5955 - AUC:
0.7388
Epoch 3/10
100/100 [==============================] - 23s 230ms/step - loss:
0.4984 - accuracy: 0.7525 - precision: 0.9386 - recall: 0.7239 - AUC:
0.8483
Epoch 4/10
100/100 [==============================] - 23s 224ms/step - loss:
0.5378 - accuracy: 0.7575 - precision: 0.8974 - recall: 0.7572 - AUC:
0.8219
Epoch 5/10
100/100 [==============================] - 21s 210ms/step - loss:
0.5117 - accuracy: 0.7600 - precision: 0.9258 - recall: 0.7421 - AUC:
0.8420
Epoch 6/10
100/100 [==============================] - 20s 204ms/step - loss:
0.6065 - accuracy: 0.7563 - precision: 0.8733 - recall: 0.7791 - AUC:
0.7747
Epoch 7/10
```

```
100/100 [==============================] - 21s 212ms/step - loss:
0.4165 - accuracy: 0.8062 - precision: 0.9488 - recall: 0.7889 - AUC:
0.8891
Epoch 8/10
100/100 [==============================] - 20s 201ms/step - loss:
0.3602 - accuracy: 0.8438 - precision: 0.9604 - recall: 0.8220 - AUC:
0.9196
Epoch 9/10
100/100 [==============================] - 21s 205ms/step - loss:
0.3409 - accuracy: 0.8450 - precision: 0.9757 - recall: 0.8111 - AUC:
0.9261
Epoch 10/10
100/100 [==============================] - 20s 202ms/step - loss:
0.3246 - accuracy: 0.8687 - precision: 0.9527 - recall: 0.8630 - AUC:
0.9362
```

29. Ewaluacja modelu.

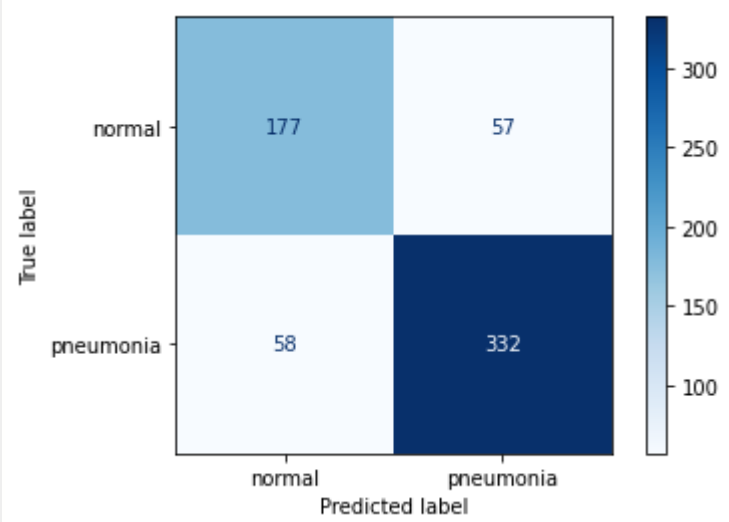
```
resnet_evaluation= resnet_model.evaluate(test)

print(f"Test Accuracy: {resnet_evaluation[1]:.2f}")
print(f"Test Precision: {resnet_evaluation[2]:.2f}")
print(f"Test Recall: {resnet_evaluation[3]:.2f}")
print(f"Test AUC: {resnet_evaluation[4]:.2f}")
```

```
624/624 [==============================] - 17s 28ms/step - loss:
0.5099 - accuracy: 0.8221 - precision: 0.8361 - recall: 0.8897 - AUC:
0.8441
Test Accuracy: 0.82
Test Precision: 0.84
Test Recall: 0.89
Test AUC: 0.84
```

30. Macierz pomyłek modelu.

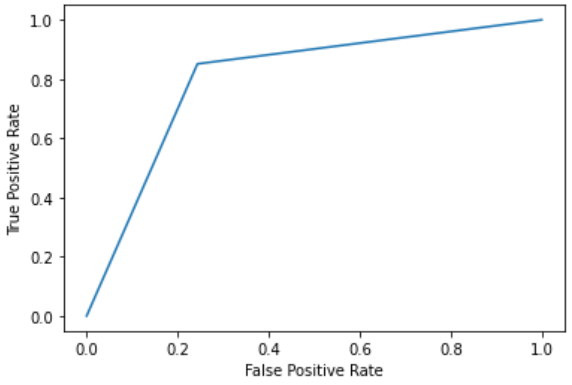
```
resnet_pred = resnet_model.predict(test)

matrix = confusion_matrix(test.classes, (resnet_pred > 0.7).ravel())

ConfusionMatrixDisplay(matrix, display_labels = ["normal",
"pneumonia"]).plot(cmap = plt.cm.Blues)
```

```
plt.show()
```



31. Wykres ROC modelu.

```
fpr, tpr, _ = metrics.roc_curve(test.classes, (resnet_pred >
0.7).ravel())

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



32. Implementacja modelu na bazie InceptionV3.

```
inceptionv3_base_model = InceptionV3(input_shape=(180,180,3),
include_top = False, weights='imagenet')

inception_model = tf.keras.Sequential([
        inceptionv3_base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(64, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(32,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")])

inception_model.compile(loss='binary_crossentropy',
                  optimizer = adam_001,
                  metrics = METRICS)
```

33. Trenowanie modelu.

```
inceprion_r = inception_model.fit(
    train,
    epochs = 10,
    validation_data = validation,
    class_weight = class_weights,
    steps_per_epoch = 100,
    validation_steps = 25)
```

```
Epoch 1/10
100/100 [==============================] - 31s 213ms/step - loss:
0.7858 - accuracy: 0.5225 - precision: 0.6835 - recall: 0.6821 - AUC:
0.5365
Epoch 2/10
100/100 [==============================] - 20s 195ms/step - loss:
0.7436 - accuracy: 0.5500 - precision: 0.7805 - recall: 0.5424 - AUC:
0.5806
```

```
Epoch 3/10
100/100 [==============================] - 20s 195ms/step - loss:
0.6305 - accuracy: 0.6075 - precision: 0.8642 - recall: 0.5582 - AUC:
0.7114
Epoch 4/10
100/100 [==============================] - 20s 197ms/step - loss:
0.5847 - accuracy: 0.6712 - precision: 0.8932 - recall: 0.6269 - AUC:
0.7669
Epoch 5/10
100/100 [==============================] - 19s 191ms/step - loss:
0.4738 - accuracy: 0.7262 - precision: 0.9465 - recall: 0.6855 - AUC:
0.8519
Epoch 6/10
100/100 [==============================] - 20s 194ms/step - loss:
0.4430 - accuracy: 0.7700 - precision: 0.9202 - recall: 0.7500 - AUC:
0.8780
Epoch 7/10
100/100 [==============================] - 20s 202ms/step - loss:
0.4004 - accuracy: 0.8037 - precision: 0.9433 - recall: 0.7755 - AUC:
0.9030
Epoch 8/10
100/100 [==============================] - 20s 198ms/step - loss:
0.4491 - accuracy: 0.7925 - precision: 0.9277 - recall: 0.7741 - AUC:
0.8748
Epoch 9/10
100/100 [==============================] - 20s 199ms/step - loss:
0.4118 - accuracy: 0.7837 - precision: 0.9468 - recall: 0.7504 - AUC:
0.8929
Epoch 10/10
100/100 [==============================] - 20s 197ms/step - loss:
0.3929 - accuracy: 0.8037 - precision: 0.9504 - recall: 0.7757 - AUC:
0.9032
```

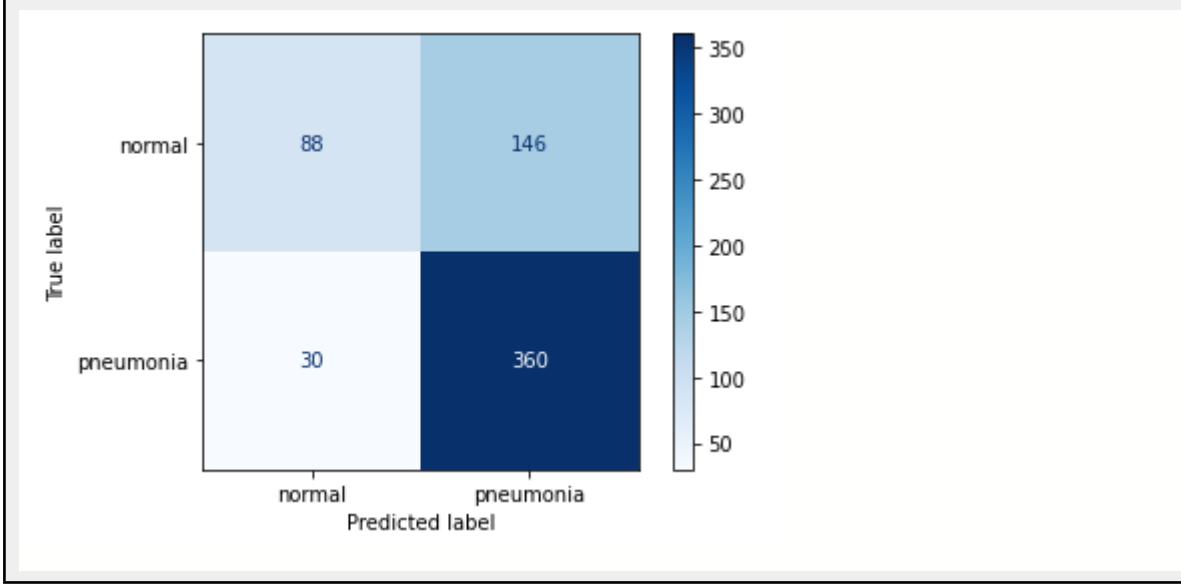
34. Ewaluacja modelu.

```
inception_evaluation = inception_model.evaluate(test)

print(f"Test Accuracy: {inception_evaluation[1]:.2f}")
print(f"Test Precision: {inception_evaluation[2]:.2f}")
print(f"Test Recall: {inception_evaluation[3]:.2f}")
print(f"Test AUC: {inception_evaluation[4]:.2f}")
```

```
624/624 [==============================] - 18s 29ms/step - loss:
17.1928 - accuracy: 0.6763 - precision: 0.6667 - recall: 0.9641 -
AUC: 0.6126
Test Accuracy: 0.68
Test Precision: 0.67
Test Recall: 0.96
Test AUC: 0.61
```

35. Macierz pomyłek modelu.

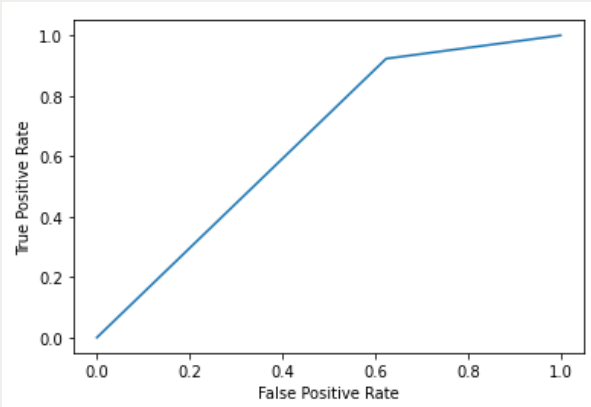
```
inception_pred = inception_model.predict(test)

matrix = confusion_matrix(test.classes, (inception_pred >
0.7).ravel())

ConfusionMatrixDisplay(matrix, display_labels = ["normal",
"pneumonia"]).plot(cmap = plt.cm.Blues)
plt.show()
```



36. Wykres ROC modelu.

```
fpr, tpr, _ = metrics.roc_curve(test.classes, (inception_pred >
0.7).ravel())

plt.plot(fpr,tpr)
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



37. Implementacja modelu na bazie EfficientNetB1.

```
efficient_base_model = EfficientNetB1(input_shape=(180,180,3),
include_top=False, weights='imagenet')

efficient_model = tf.keras.Sequential([
        efficient_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.2),
        Dense(1,activation="sigmoid")])

efficient_model.compile(loss='binary_crossentropy',
                  optimizer = adam_001,
                  metrics = METRICS)
```

38. Trenowanie modelu.

```
efficient_r = efficient_model.fit(
```

```
    train,
    epochs = 10,
    validation_data = validation,
    class_weight = class_weights,
    steps_per_epoch = 100,
    validation_steps = 25)
```

```
Epoch 1/10
100/100 [==============================] - 37s 239ms/step - loss:
0.6970 - accuracy: 0.6662 - precision: 0.7144 - recall: 0.8064 - AUC:
0.5392
Epoch 2/10
100/100 [==============================] - 21s 212ms/step - loss:
0.5039 - accuracy: 0.7775 - precision: 0.9256 - recall: 0.7716 - AUC:
0.8386
Epoch 3/10
100/100 [==============================] - 21s 214ms/step - loss:
0.4159 - accuracy: 0.7887 - precision: 0.9509 - recall: 0.7623 - AUC:
0.8888
Epoch 4/10
100/100 [==============================] - 22s 215ms/step - loss:
0.3943 - accuracy: 0.8000 - precision: 0.9514 - recall: 0.7756 - AUC:
0.8983
Epoch 5/10
100/100 [==============================] - 22s 214ms/step - loss:
0.3555 - accuracy: 0.8562 - precision: 0.9502 - recall: 0.8541 - AUC:
0.9190
Epoch 6/10
100/100 [==============================] - 21s 213ms/step - loss:
0.3406 - accuracy: 0.8462 - precision: 0.9631 - recall: 0.8267 - AUC:
0.9253
Epoch 7/10
100/100 [==============================] - 22s 218ms/step - loss:
0.2997 - accuracy: 0.8612 - precision: 0.9619 - recall: 0.8473 - AUC:
0.9447
Epoch 8/10
100/100 [==============================] - 21s 212ms/step - loss:
0.2842 - accuracy: 0.8825 - precision: 0.9649 - recall: 0.8814 - AUC:
0.9463
Epoch 9/10
100/100 [==============================] - 22s 214ms/step - loss:
0.2985 - accuracy: 0.8687 - precision: 0.9551 - recall: 0.8629 - AUC:
0.9432
```

```
Epoch 10/10
100/100 [==============================] - 21s 210ms/step - loss:
0.3036 - accuracy: 0.8913 - precision: 0.9572 - recall: 0.8950 - AUC:
0.9402
```

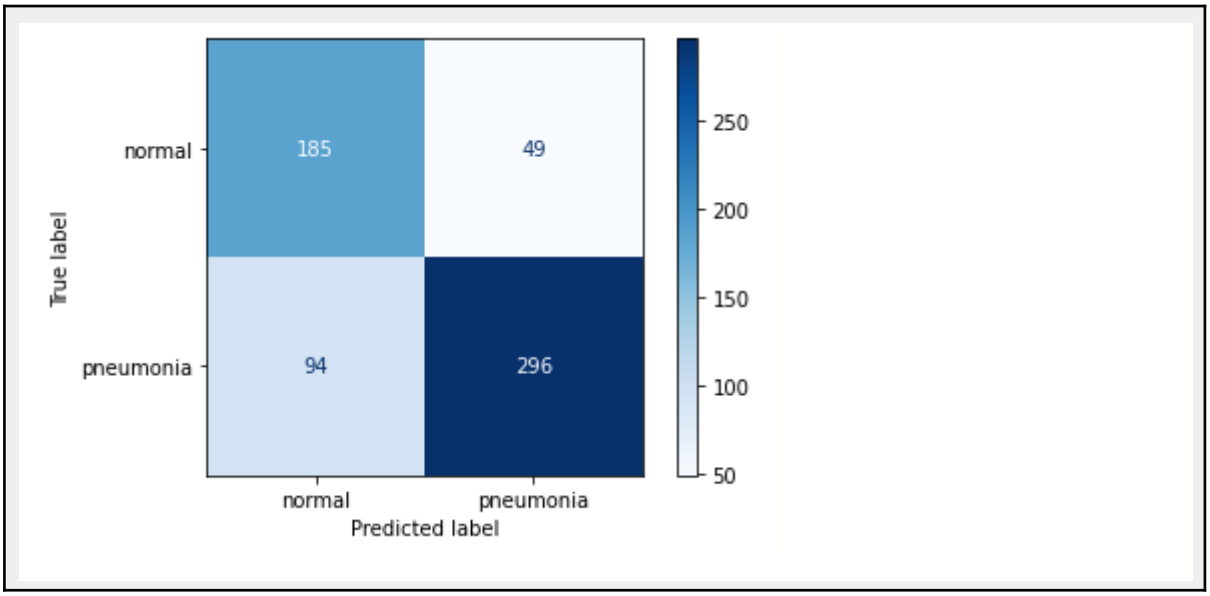
39. Ewaluacja modelu.

```
efficient_evaluation = efficient_model.evaluate(test)

print(f"Test Accuracy: {efficient_evaluation[1]:.2f}")
print(f"Test Precision: {efficient_evaluation[2]:.2f}")
print(f"Test Recall: {efficient_evaluation[3]:.2f}")
print(f"Test AUC: {efficient_evaluation[4]:.2f}")
```

```
624/624 [==============================] - 19s 30ms/step - loss:
0.4638 - accuracy: 0.7804 - precision: 0.7674 - recall: 0.9308 - AUC:
0.8561
Test Accuracy: 0.78
Test Precision: 0.77
Test Recall: 0.93
Test AUC: 0.86
```

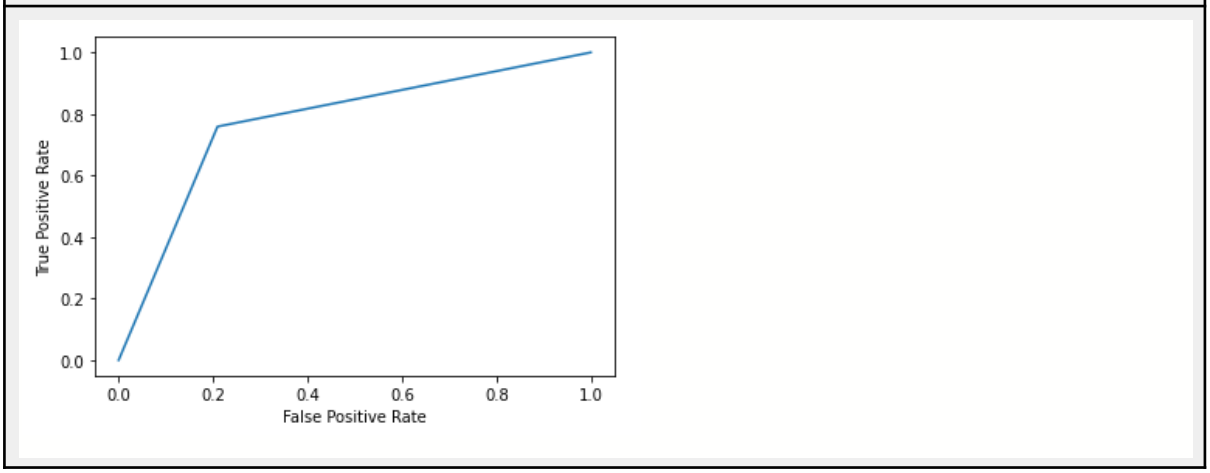
40. Macierz pomyłek modelu.

```
efficient_pred = eficient_model.predict(test)

matrix = confusion_matrix(test.classes, (efficient_pred >
0.7).ravel())

ConfusionMatrixDisplay(matrix, display_labels = ["normal",
"pneumonia"]).plot(cmap = plt.cm.Blues)
plt.show()
```



41. Wykres ROC modelu.

```
fpr, tpr, _ = metrics.roc_curve(test.classes, (efficient_pred >
0.7).ravel())

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



6. Ewaluacja wyników.

Do przeprowadzenia ewaluacji wyników uzyskanych po treningu sieci wykorzystano wbudowaną metodę *evaluate* dla klasy *tf.keras.Model()*. Pozwala ona na obliczenia wcześniej zdefiniowanych metryk. Przeprowadzono analizę na zbiorze testowym, dla którego otrzymano następujące parametry:

- **Accuracy** - ile przypadków zostało prawidłowo zaklasyfikowanych, zarówno jako pozytywne, jak i negatywne.

$$\frac{TP + TN}{TP + TN + FN + FP}$$

- **Recall** - ile pozytywnych przypadków zostało wykrytych ze wszystkich naprawdę pozytywnych.

$$\frac{TP}{TP + FN}$$

- **Precision** - ile przypadków z uznanych przez klasyfikator za pozytywne jest w rzeczywistości pozytywnych.

$$\frac{TP}{TP + FP}$$

- **F1 Score** - średnia harmoniczna parametrów precision i recall. Im bliższy 1, tym lepiej radzi sobie klasyfikator.

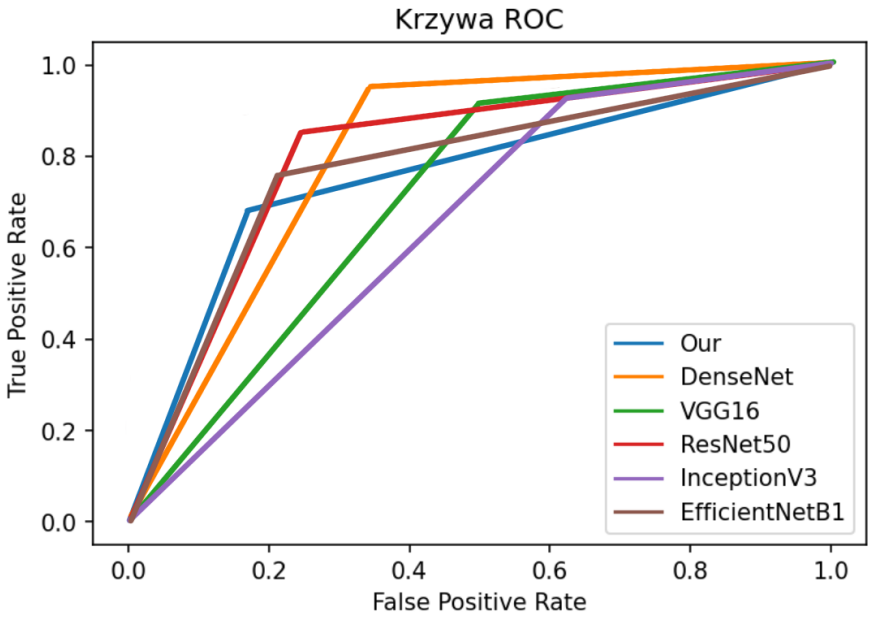
$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- **AUC** - powierzchnia pod krzywą ROC.

Tabela 1. Zestawienie otrzymanych metryk dla wszystkich typów sieci.

Typ sieci konwolucyjnej	Wybrane parametry				
	Accuracy	Recall	Precision	F1 Score	AUC
Implementacja własna	0,85	0,91	0,86	0,88	0,89
Bazująca na DenseNet121	0,84	0,95	0,83	0,89	0,92
Bazująca na VGG16	0,76	0,91	0,76	0,83	0,86
Bazująca na ResNet50	0,82	0,89	0,84	0,86	0,84
Bazująca na InceptionV3	0,68	0,96	0,67	0,79	0,61
Bazująca na EfficientNetB1	0,78	0,93	0,77	0,84	0,86

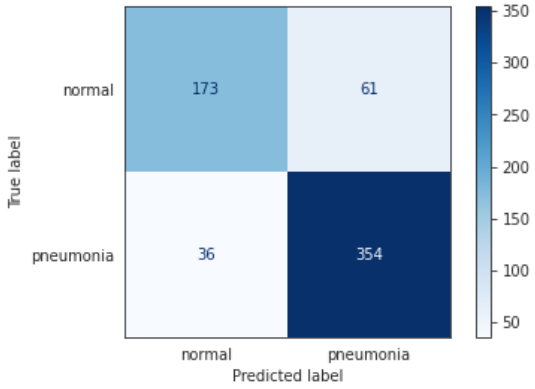
Istotny jest również wykres ROC (Receiver Operating Characteristic). Jest to graficzna ocena jakości klasyfikatora binarnego. Pozwala na jednoczesny opis jego czułości i specyficzności. Otrzymane krzywe dla każdego modelu zestawiono na jednym grafie. (Rys. 6.1)



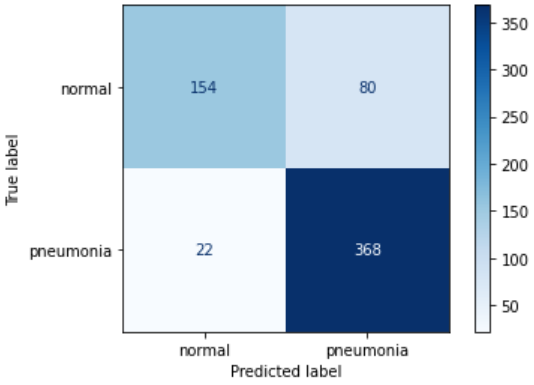
Rys. 6.1. Wykresy ROC dla poszczególnych modeli.

Kolejnym typem ewaluacji była ocena macierzy pomyłek otrzymanych dla poszczególnym typów sieci. Jest to również graficzny typ oceny klasyfikacji binarnej.

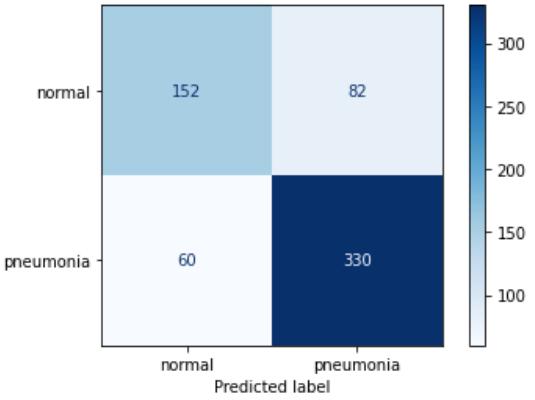
Tablica ma dwa wiersze i dwie kolumny, gdzie wiersze przedstawiają klasy przewidywane, kolumny zaś klasy rzeczywiste. Przy pomocy biblioteki *sklearn.metrics* zwizualizowano tablice dodając kolory w celu większej przejrzystości (Rys. 6.2. - 6.7.)



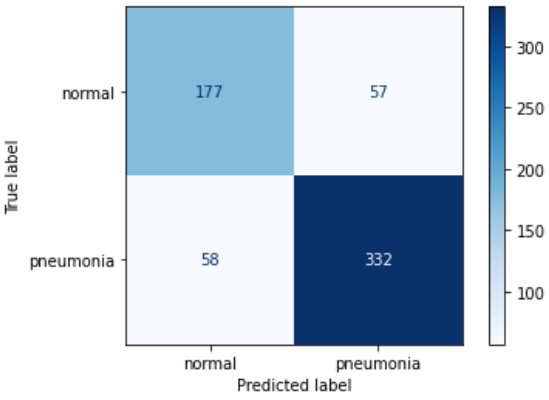
Rys 6.2. Macierz pomyłek dla sieci implementacji własnej.



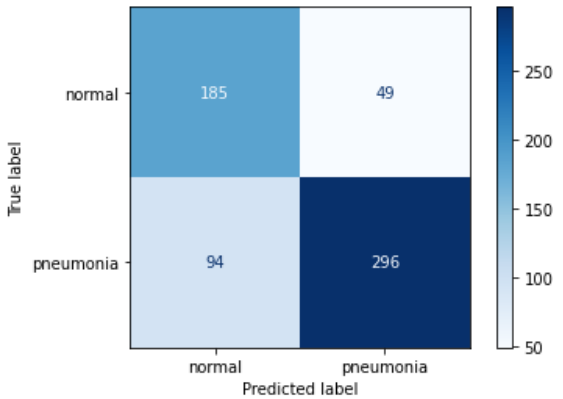
Rys 6.3. Macierz pomyłek dla sieci bazującej na DenseNet.

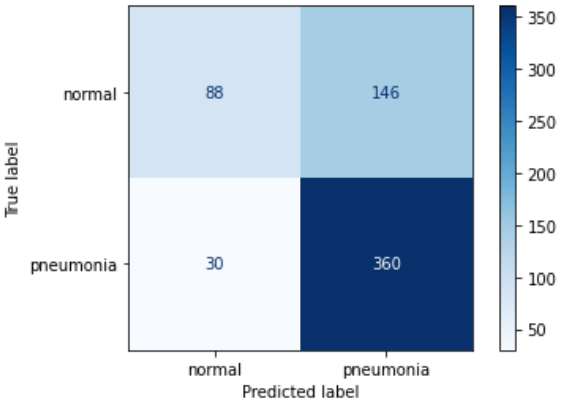


Rys 6.4. Macierz pomyłek dla sieci bazującej na VGG16.



Rys 6.5. Macierz pomyłek dla sieci bazującej na ResNet50.





Rys 6.6. Macierz pomyłek dla sieci bazującej na InceptionV3.

Rys 6.7. Macierz pomyłek dla sieci bazującej na EfficientNetB1.

7. Dyskusja.

Otrzymane wartości metryk dla wszystkich klasyfikatorów znajdują się na zadowalającym poziomie powyżej lub blisko 70%. Oznacza to, że każdy z nich poradził sobie z zadaniem klasyfikacji na tyle wystarczająco, aby uznać je za zdatne do dalszej analizy. Spośród nich najwyższy poziom metryki Accuracy osiągnęła sieć naszej implementacji - otrzymana wartość to 85%. Jest to prawdopodobnie związane z tym że została ona najlepiej dopasowana do pracy na wybranym zbiorze zdjęć RTG. Równie dobrze poradziły sobie modele bazujące na sieci DenseNet jak i ResNet50 z wynikami 84% i 82% odpowiednio. Jednak oprócz dokładności, bardzo istotna jest wartości metryki Recall. W przypadku klasyfikacji jednostek chorobowych, najistotniejsze jest poprawne wytypowanie przypadków wystąpienia patologii, o czym mówi właśnie ten parametr. Najwyższą wartość uzyskał model sieci InceptionV3, ale biorąc pod uwagę niską wartość Accuracy jak i AUC nie możemy uznać go za najlepszy. Pod tym względem najlepiej poradziła sobie sieć ResNet50. Dodatkowo posiada ona najwyższą wartość metryki AUC 92% oraz F1 Score 89%.

Warto nadmienić, że podczas trenowania wszystkich modeli zauważono wyższe wartości metryk dla zbioru treningowego, niż później otrzymane dla testowego. Oznacza to, że pojawił się problem Overfitting'u, czyli nadmiernego dopasowania. Zagadnienie to stanowi duży problem, gdyż ogranicza szersze zastosowanie proponowanych modeli w celach diagnostycznych. Jego wystąpienie jest również dość zaskakujące, biorąc pod uwagę wykonane transformacje samych zdjęć, jak i warstwy w sieciach mające na celu zniwelowanie tego zjawiska np. Dropout czy BatchNormalization. Nie mniej, mając na uwadze dalszy rozwój przedstawionych modeli, proponowane byłoby podjęcie większej ilości kroków w przeciwdziałaniu temu zjawisku.

8. Podsumowanie i wnioski.

Prawidłowa diagnostyka zapalenia płuc jest bardzo istotnym zagadnieniem, szczególnie w przypadku dzieci w wieku poniżej 5 lat. Tak jak było na samym początku wspomniane, jest to najczęstsza przyczyna zgonów wśród omawianej grupy wiekowej. Dlatego aspekt wykorzystania uczenia maszynowego w celu ułatwienia tego procesu jest ciągle badanym i rozwijanym problemem. W przedstawionym przez nas projekcie zaproponowano 6 różnych modeli bazujących na sieciach konwolucyjnych wcześniej przetrenowanych jak i implementacji własnej. Spośród nich wytypowano sieć DenseNet121 jako najskuteczniejszy w klasyfikacji badanego zbioru danych. Wykazała ona wysokie wartości metryk takich jak: Accuracy (84%), Recall (95%), Precision (83%),

F1 Score (89%), AUC (95%). Najgorzej natomiast przeprowadzona została klasyfikacja na bazie sieci InceptionV3, dla której wartość Accuracy, Precision jak i AUC były na poziomie poniżej 70%. Interesującym faktem jest to, że sieć implementacji własnej miała podobne wartości metryk co DenseNet121 czy ResNet50, pomimo mniejszej liczby warstw. Pokazuje to jak istotne jest dopasowanie architektury sieci do badanego zagadnienia. Dlatego też pomimo tego, że sieci InceptionV3 i EfficientNetB1 są nowszej generacji, nie sprawdziły się tak dobrze, gdyż wymagają bardziej złożonego dopasowania. Jako perspektywy rozwoju projektu można by założyć bardziej złożona transformację zdjęć, jak również rozważenie innych sposobów równoważenia zbioru. Korzystne byłoby też poszerzenie badanego zbioru o zdjęcia z innych szpitali. Proponowane algorytmy nie są idealne, ich implementacja wymagałaby ulepszenia, w celu zwiększenia wartości otrzymanych metryk. Są one natomiast obiecujące pod względem możliwości uczenia maszynowego.

9. Bibliografia.

[1] „Zapalenie płuc”. //www.poradnikzdrowie.pl/zdrowie/uklad-oddechowy/zapalenie-pluc-przyczyny-rodzaje-objawy-powiklania-aa-b1fT-nv32-yY3p.html (dostęp 28 maj 2022).

[2] „Pediatria po Dyplomie - Zapalenie płuc u dzieci – do kiedy można leczyć w domu”. https://podyplomie.pl/pediatria/24534,zapalenie-pluc-u-dzieci-do-kiedy-mozna-leczyc-w-domu (dostęp 29 maj 2022).

[10] https://www-1sciencedirect-1com-10000273801c4.wbg2.bg.agh.edu.pl/science/article/pii/S0957417420310423

[11] https://en.wikipedia.org/wiki/Convolutional_neural_network

[Zbiór danych]https://www.kaggle.com/datasets/pcbreviglieri/pneumonia-xray-images.