Raport - Wykrywanie samochodów na zdjęciach za pomocą konwolucyjnej sieci neuronowej (CNN)

Chylak Michał 97176

Naglak Tomasz 98821

Spis treści

[Cel zadania 3](#_Toc169046518)

[Przygotowania 3](#_Toc169046519)

[Krok 1: Wstępne przetwarzanie danych 3](#_Toc169046520)

[Krok 2: Trenowanie konwolucyjnej sieci neuronowej 3](#_Toc169046521)

[Krok 3: Rozszerzanie danych 4](#_Toc169046522)

[Krok 4: Wykorzystanie wcześniej wytrenowanego modelu 4](#_Toc169046523)

[Krok 5: Wizualizacja wyuczonych cech 4](#_Toc169046524)

[Opis kodu 5](#_Toc169046525)

[1. Importowanie bibliotek 5](#_Toc169046526)

[2. Wczytanie i przeskalowanie obrazów z katalogu treningowego 5](#_Toc169046527)

[3. Definicja modelu VGG-16 6](#_Toc169046528)

[4. Trening modelu VGG-16 6](#_Toc169046529)

[5. Ocena modelu VGG-16 6](#_Toc169046530)

[6. Definicja modelu CNN 7](#_Toc169046531)

[7. Przygotowanie klasy zapisującej epokę, w której powstał najlepszy model 7](#_Toc169046532)

[8. Trenowanie modelu CNN 7](#_Toc169046533)

[9. Ocena modelu na danych walidacyjnych 8](#_Toc169046534)

[10. Wizualizacja historii treningu 8](#_Toc169046535)

[11. Ekstrakcja cech z obrazów przy użyciu VGG-16 9](#_Toc169046536)

[12. Klasyfikator regresji logistycznej 9](#_Toc169046537)

[13. Wizualizacja wyekstrahowanych cech przy użyciu t-SNE 9](#_Toc169046538)

[14. Wizualizacja warstw konwolucyjnych 10](#_Toc169046539)

[Wizualizacje 10](#_Toc169046540)

[Wizualizacja cech 1 warstwy modelu własnego i VGG16 10](#_Toc169046541)

[Dokładność i strata 11](#_Toc169046542)

[Dokładność i strata po 5 epokach 11](#_Toc169046543)

[Dokładność i strata po 20 epokach 11](#_Toc169046544)

[Dokładność i strata po 50 epokach 12](#_Toc169046545)

[Wizualizacja ekstraktora cech 12](#_Toc169046546)

[Podsumowanie 13](#_Toc169046547)

[Literatura i źródła 13](#_Toc169046548)

# Cel zadania

Celem zadania jest praktyczne zastosowanie konwolucyjnych sieci neuronowych do klasyfikacji obrazów. Model CNN, który ma za zadanie rozpoznawanie obrazów przedstawiających samochody.

# Przygotowania

*1. Zapoznanie się z materiałami o CNN.  
2. Instalacja potrzebnych bibliotek: Zainstaluj Keras i TensorFlow za pomocą Anaconda lub pip. Dodatkowo, zainstaluj bibliotekę Pillow do przetwarzania obrazów.  
3. Wykorzystanie wydajnego sprzętu: Zalecane jest użycie komputera z dostępem do GPU.*

Zapoznaliśmy się z załączoną w zadaniu literaturą oraz innymi źródłami dostępnymi w internecie. Zdecydowaliśmy się na pracę nad zadaniem lokalnie na naszych komputerach. Kod utrzymywany był za pomoca narzędzia git i utworzonego na potrzeby zadania repozytorium. Nie udało nam się skonfigurować środowiska w taki sposób by napisany przez nas program korzystał z GPU, ale moc CPU pozwalała uczenie modelu w zadowalającym dla nas tempie (szybszym niż za pomocą notatnika Google Colab).

Link do repozytorium git: <https://github.com/Michauue/machine_learning/tree/final>

# Krok 1: Wstępne przetwarzanie danych

*Ładowanie danych: Użyj ImageDataGenerator z Keras do wczytania i przeskalowania obrazów ze zbioru danych PASCAL VOC 2010 lub podobnego. Parametry generatora powinny obejmować m.in. skalowanie wartości pikseli, rozmiar obrazu, wielkość partii, tryb klasyfikacji oraz losowość próbek.*

Ten etap projektu nie był wymagający, polegał tylko na zastosowaniu wskazanego w poleceniu narzędzia ImageDataGenerator.

# Krok 2: Trenowanie konwolucyjnej sieci neuronowej

*Definicja modelu CNN: Napisz funkcję make\_convnet tworzącą model CNN.  
Trenowanie modelu: Użyj metody fit na przygotowanym modelu z użyciem wygenerowanych danych treningowych i walidacyjnych.  
Zapis i ocena modelu: Zapisz wyuczone wagi modelu i ocen jego dokładność.*

W tej części zadania zaimplementowaliśmy model zgodnie ze wskazówkami jakie udało nam się znaleźć w różnych źródłach. Zaproponowana konfiguracja warstw, funkcji aktywacji oraz optymalizatora osiągnęła po przeprowadzonych testach i porównaniach najlepsze wyniki. Finalny model został przez nas zapisany, by być łatwo i szybko dostępnym do zaimplementowania, bez konieczności przechodzenia przez proces uczenia kolejny raz.

# Krok 3: Rozszerzanie danych

*Eksperymenty z augmentacją danych: Stwórz nowy ImageDataGenerator do zwiększenia różnorodności danych treningowych przez transformacje obrazu.  
Ponowne trenowanie i analiza wyników: Trenuj model z nowymi danymi i analizuj dokładność oraz straty dla kolejnych epok.*

Podczas procesu uczenia zauważyliśmy związek między ilością epok, a finalną skutecznością modelu – im większa ilość epok, tym lepsze wyniki osiągał model. Zwiększenie różnorodności danych treningowych w naszych obserwacjach nie wpływało znacząco na poprawę modelu. Na tym etapie zadania postanowiliśmy zaimplementować funkcję zapisującą z epoki o jakim numerze pochodzi model mający najlepsze wyniki. Zdecydowaliśmy się, by najlepszy model determinowany był przez najwyższą wartość dokładności na danych walidacyjnych.

# Krok 4: Wykorzystanie wcześniej wytrenowanego modelu

*Użycie modelu VGG-16: Załaduj wytrenowany model VGG-16, przeanalizuj możliwości jego zastosowania jako klasyfikatora oraz jako ekstraktora cech.  
Eksperymenty z transfer learning: Zastosuj ekstrakcję cech przy użyciu VGG-16 i trenuj własny klasyfikator na tych danych.*

Model VGG-16 został zaimplementowany za pomocą biblioteki keras.applications.vgg16. W dalszej części dokumentu przedstawiamy porównanie wyników tego modelu z wynikami naszego własnego modelu CNN. Analiza wyników pokazuje, że model VGG-16 osiągnął wyższą dokładność oraz charakteryzował się niższą wartością funkcji straty w porównaniu z naszym modelem. Ponadto, linie wykresu dokładności dla danych treningowych i walidacyjnych VGG-16 są blisko siebie, co świadczy o jego lepszym dopasowaniu i zdolności do generalizacji. Oznacza to, że model VGG-16 dobrze przewiduje zarówno na zbiorze treningowym, jak i walidacyjnym, bez oznak nadmiernego dopasowania (overfittingu) ani niedopasowania (underfittingu). Jednocześnie, nasz model również wykazuje dobre dopasowanie, choć nie osiąga tak wysokiej precyzji jak VGG-16. Linie wykresu dla naszego modelu są także blisko siebie, co sugeruje, że nasz model jest stabilny i zdolny do generalizacji, mimo że jego architektura i parametry mogą wymagać dalszej optymalizacji, aby dorównać modelowi VGG-16.

# Krok 5: Wizualizacja wyuczonych cech

*Analiza pierwszej warstwy splotowej: Zwizualizuj cechy wyuczone przez pierwszą warstwę modelu VGG-16 oraz własnego modelu CNN.*

Wizualizacja cech wyuczonych przez oba modele jest podobna, jednak w przypadku VGG-16 dwukrotnie większa ilość filtrów w pierwszej warstwie skutkuje większą liczbą “klatek” na wizualizacji. Obrazy uzyskane przez nas podczas realizacji kroku przedstawiamy w dalszej części dokumentu.

# Opis kodu

## 1. Importowanie bibliotek

Kod zaczyna się od importowania potrzebnych bibliotek, takich jak matplotlib, keras, sklearn oraz numpy.

import matplotlib.pyplot as plt

from keras.applications.vgg16 import VGG16

from keras.models import Model, Sequential

from keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D

from keras.preprocessing.image import ImageDataGenerator

from keras.optimizers import Adam

from keras.callbacks import ModelCheckpoint, Callback

import numpy as np

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make\_pipeline

from sklearn.metrics import accuracy\_score

from sklearn.manifold import TSNE

## 2. Wczytanie i przeskalowanie obrazów z katalogu treningowego

Stworzenie instancji ImageDataGenerator z różnymi augmentacjami obrazu i przygotowanie generatorów dla zbiorów treningowego i walidacyjnego.

datagen = ImageDataGenerator(

    rescale=1.0/255,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    vertical\_flip=True,

    rotation\_range=40,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    brightness\_range=[0.8, 1.2],

    channel\_shift\_range=0.2

)

train\_generator = datagen.flow\_from\_directory(

    'CNN\_zadanie\_images/images/train',

    target\_size=(150, 150),

    batch\_size=32,

    class\_mode='binary',

    shuffle=True

)

validation\_generator = datagen.flow\_from\_directory(

    'CNN\_zadanie\_images/images/validation',

    target\_size=(150, 150),

    batch\_size=32,

    class\_mode='binary',

    shuffle=True

)

## 3. Definicja modelu VGG-16

Funkcja create\_vgg16 definiuje model VGG-16 bez górnych warstw, dodaje własne warstwy na szczycie i kompiluje model. Definiujemy także ilość epok przez którą przprowadzane będzie późniejsze uczenie modelu.

xepochs = 50

def create\_vgg16(input\_shape):

    base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    x = base\_model.output

    x = Flatten()(x)

    x = Dense(512, activation='relu')(x)

    x = Dropout(0.5)(x)

    predictions = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=base\_model.input, outputs=predictions)

    for layer in base\_model.layers:

        layer.trainable = False

    model.compile(optimizer=Adam(), loss='binary\_crossentropy', metrics=['accuracy'])

    return model

input\_shape = (150, 150, 3)

vgg16\_model = create\_vgg16(input\_shape)

## 4. Trening modelu VGG-16

Model jest trenowany na danych treningowych przy użyciu metod fit i ModelCheckpoint do zapisu najlepszego modelu.

checkpoint\_path\_vgg16 = "best\_model\_vgg16.h5"

checkpoint\_vgg16 = ModelCheckpoint(checkpoint\_path\_vgg16, monitor='val\_accuracy', save\_best\_only=True, mode='max')

history\_vgg16 = vgg16\_model.fit(

    train\_generator,

    steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

    validation\_data=validation\_generator,

    validation\_steps=validation\_generator.samples // validation\_generator.batch\_size,

    epochs=xepochs,

    callbacks=[checkpoint\_vgg16]

)

vgg16\_model.load\_weights(checkpoint\_path\_vgg16)

## 5. Ocena modelu VGG-16

Załadowanie najlepszych wag modelu VGG-16 i ocena na zbiorze walidacyjnym.

loss\_vgg16, accuracy\_vgg16 = vgg16\_model.evaluate(validation\_generator)

print(f'VGG16 model loss: {loss\_vgg16}')

print(f'VGG16 model accuracy: {accuracy\_vgg16}')

## 6. Definicja modelu CNN

Funkcja make\_convnet tworzy zaproponowany własny model konwolucyjnej sieci neuronowej.

def make\_convnet(input\_shape):

    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(512, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=Adam(), loss='binary\_crossentropy', metrics=['accuracy'])

    return model

model = make\_convnet(input\_shape)

## 7. Przygotowanie klasy zapisującej epokę, w której powstał najlepszy model

Klasa EpochSaver, która zapisuje najlepszy model na podstawie walidacji (inwencja własna, nie wymagana w zadaniu).

class EpochSaver(Callback):

    def \_\_init\_\_(self):

        super(EpochSaver, self).\_\_init\_\_()

        self.best\_epoch = 0

        self.best\_val\_accuracy = -np.Inf

    def on\_epoch\_end(self, epoch, logs=None):

        current\_val\_accuracy = logs.get('val\_accuracy')

        if current\_val\_accuracy > self.best\_val\_accuracy:

            self.best\_val\_accuracy = current\_val\_accuracy

            self.best\_epoch = epoch + 1

    def on\_train\_end(self, logs=None):

        print(f'Best model saved from epoch {self.best\_epoch} with val\_accuracy {self.best\_val\_accuracy:.4f}')

## 8. Trenowanie modelu CNN

Model jest trenowany na danych treningowych przy użyciu metod fit, ModelCheckpoint oraz EpochSaver.

checkpoint\_path = "best\_model.h5"

checkpoint = ModelCheckpoint(checkpoint\_path, monitor='val\_accuracy', save\_best\_only=True, mode='max')

epoch\_saver = EpochSaver()

history = model.fit(

    train\_generator,

    steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

    validation\_data=validation\_generator,

    validation\_steps=validation\_generator.samples // validation\_generator.batch\_size,

    epochs=xepochs,

    callbacks=[checkpoint, epoch\_saver]

)

## 9. Ocena modelu na danych walidacyjnych

Ocena własnego modelu na danych walidacyjnych i zapis wyuczonych wag modelu.

val\_loss, val\_accuracy = model.evaluate(validation\_generator, steps=validation\_generator.samples // validation\_generator.batch\_size)

print(f'Validation loss: {val\_loss}')

print(f'Validation accuracy: {val\_accuracy}')

model.load\_weights(checkpoint\_path)

model.save('final\_model.h5')

loss, accuracy = model.evaluate(validation\_generator)

print(f'Final model loss: {loss}')

print(f'Final model accuracy: {accuracy}')

## 10. Wizualizacja historii treningu

Funkcja plot\_history tworzy wykresy dokładności i strat dla modelu VGG-16 i własnego modelu.

def plot\_history(history1, history2, model1\_name, model2\_name):

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)

    plt.plot(history1.history['accuracy'], label=f'{model1\_name} Training Accuracy')

    plt.plot(history1.history['val\_accuracy'], label=f'{model1\_name} Validation Accuracy')

    plt.plot(history2.history['accuracy'], label=f'{model2\_name} Training Accuracy')

    plt.plot(history2.history['val\_accuracy'], label=f'{model2\_name} Validation Accuracy')

    plt.title('Training and Validation Accuracy')

    plt.xlabel('Epochs')

    plt.ylabel('Accuracy')

    plt.legend()

    plt.subplot(1, 2, 2)

    plt.plot(history1.history['loss'], label=f'{model1\_name} Training Loss')

    plt.plot(history1.history['val\_loss'], label=f'{model1\_name} Validation Loss')

    plt.plot(history2.history['loss'], label=f'{model2\_name} Training Loss')

    plt.plot(history2.history['val\_loss'], label=f'{model2\_name} Validation Loss')

    plt.title('Training and Validation Loss')

    plt.xlabel('Epochs')

    plt.ylabel('Loss')

    plt.legend()

    plt.tight\_layout()

    plt.show()

plot\_history(history, history\_vgg16, 'My Model', 'VGG16')

## 11. Ekstrakcja cech z obrazów przy użyciu VGG-16

Funkcja extract\_features używa warstw konwolucyjnych VGG-16 do ekstrakcji cech z obrazów.

def extract\_features(model, generator, steps):

    features = []

    labels = []

    for i in range(steps):

        x\_batch, y\_batch = next(generator)

        features\_batch = model.predict(x\_batch)

        features.append(features\_batch)

        labels.append(y\_batch)

    return np.vstack(features), np.hstack(labels)

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=input\_shape)

feature\_extractor = Model(inputs=base\_model.input, outputs=base\_model.get\_layer('block5\_pool').output)

train\_features, train\_labels = extract\_features(feature\_extractor, train\_generator, train\_generator.samples // train\_generator.batch\_size)

validation\_features, validation\_labels = extract\_features(feature\_extractor, validation\_generator, validation\_generator.samples // validation\_generator.batch\_size)

## 12. Klasyfikator regresji logistycznej

Utworzenie klasyfikatora przy użyciu regresji logistycznej, trenowanie na wyekstrahowanych cechach i ocena na danych walidacyjnych.

classifier = make\_pipeline(StandardScaler(), LogisticRegression(max\_iter=10000))

classifier.fit(train\_features.reshape(train\_features.shape[0], -1), train\_labels)

val\_predictions = classifier.predict(validation\_features.reshape(validation\_features.shape[0], -1))

accuracy = accuracy\_score(validation\_labels, val\_predictions)

print(f'Feature extractor + Logistic Regression accuracy: {accuracy}')

## 13. Wizualizacja wyekstrahowanych cech przy użyciu t-SNE

Funkcja plot\_tsne wizualizuje wyekstrahowane cechy przy użyciu t-SNE.

def plot\_tsne(features, labels, title):

    tsne = TSNE(n\_components=2, random\_state=0)

    features\_2d = tsne.fit\_transform(features)

    plt.figure(figsize=(8, 8))

    for label in np.unique(labels):

        indices = np.where(labels == label)

        plt.scatter(features\_2d[indices, 0], features\_2d[indices, 1], label=f'Class {label}')

    plt.legend()

    plt.title(title)

    plt.show()

plot\_tsne(train\_features.reshape(train\_features.shape[0], -1), train\_labels, 'VGG16 Features (Train)')

plot\_tsne(validation\_features.reshape(validation\_features.shape[0], -1), validation\_labels, 'VGG16 Features (Validation)')

## 14. Wizualizacja warstw konwolucyjnych

Funkcja visualize\_conv\_layer wizualizuje wyuczone cechy pierwszej warstwy splotowej dla modelu VGG-16 oraz własnego modelu.

def visualize\_conv\_layer(model, layer\_name, image):

    intermediate\_layer\_model = Model(inputs=model.input,

                                     outputs=model.get\_layer(layer\_name).output)

    intermediate\_output = intermediate\_layer\_model.predict(image)

    num\_filters = intermediate\_output.shape[-1]

    size = intermediate\_output.shape[1]

    display\_grid = np.zeros((size, size \* num\_filters))

    for i in range(num\_filters):

        x = intermediate\_output[0, :, :, i]

        x -= x.mean()

        x /= (x.std() + 1e-5)

        x \*= 64

        x += 128

        x = np.clip(x, 0, 255).astype('uint8')

        display\_grid[:, i \* size : (i + 1) \* size] = x

    scale = 20. / num\_filters

    plt.figure(figsize=(scale \* num\_filters, scale))

    plt.title(layer\_name)

    plt.grid(False)

    plt.imshow(display\_grid, aspect='auto', cmap='viridis')

sample\_image = next(train\_generator)[0][0]

sample\_image = np.expand\_dims(sample\_image, axis=0)

visualize\_conv\_layer(model, 'conv2d', sample\_image)

visualize\_conv\_layer(vgg16\_model, 'block1\_conv1', sample\_image)

plt.show()

# Wizualizacje

## Wizualizacja cech 1 warstwy modelu własnego i VGG16

Poniżej znajdują się wizualizacje cech wyuczonych przez pierwszą warstwę modelu własnego oraz modelu VGG-16.

Wizualizacja cech pierwszej warstwy modelu własnego:

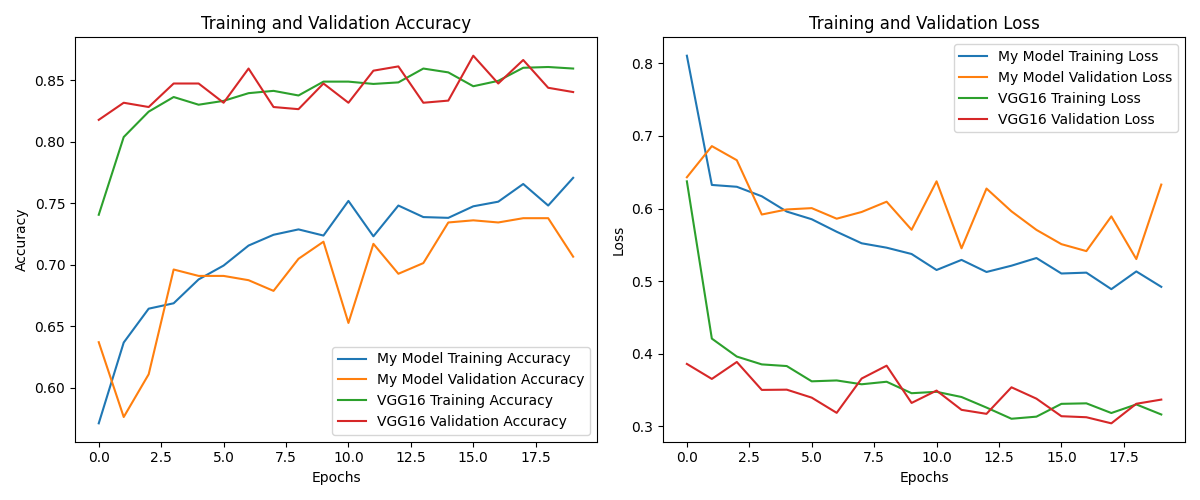
Wizualizacja cech pierwszej warstwy modelu VGG-16:

## Dokładność i strata

Poniżej znajdują się wykresy dokładności i straty dla autorskiego modelu CNN oraz VGG-16 po 5, 20 i 50 epokach:

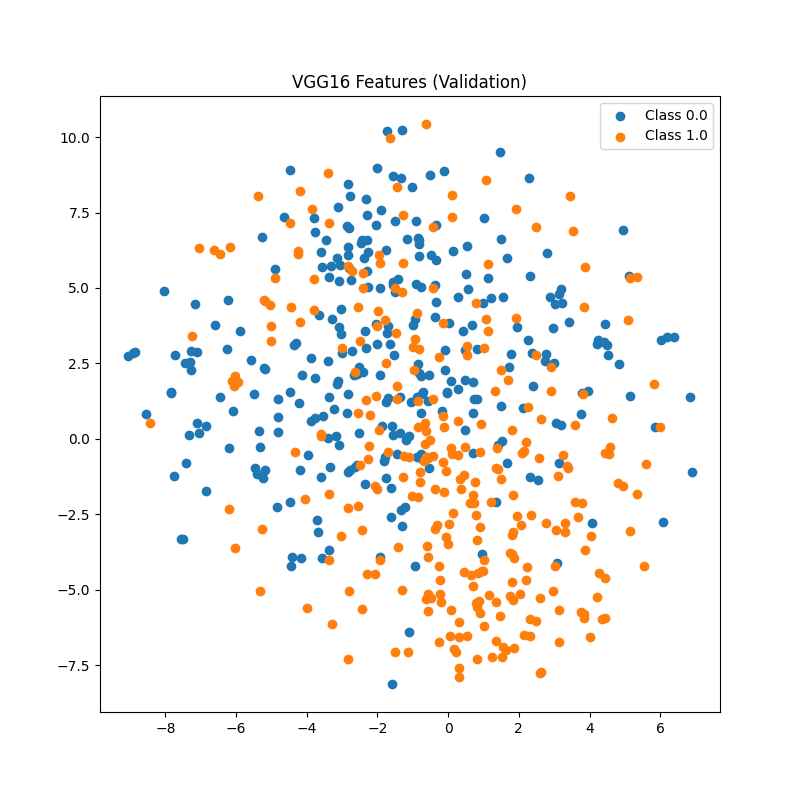
### Dokładność i strata po 5 epokach

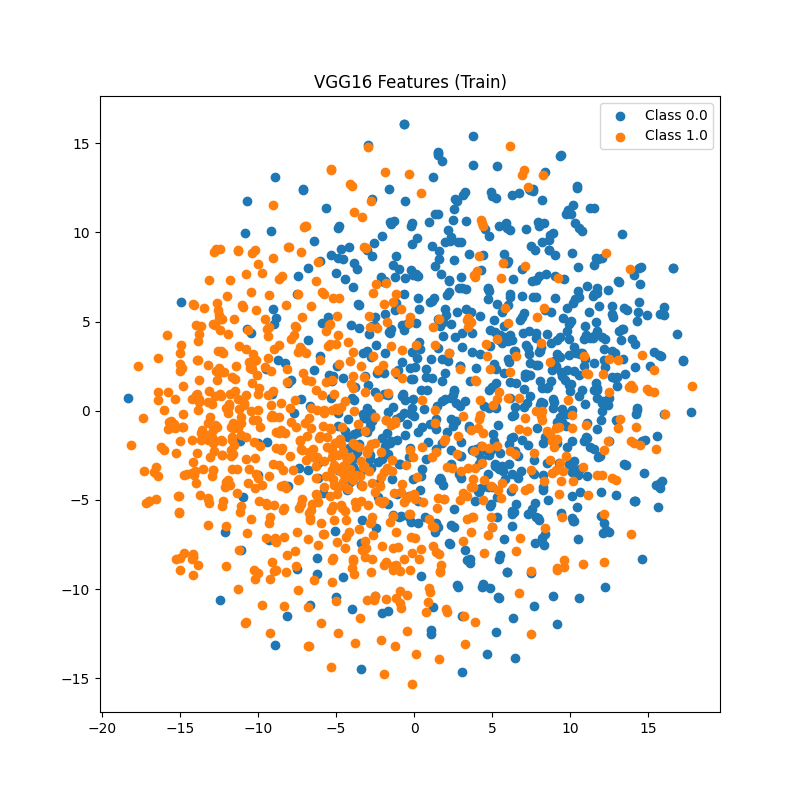
### Dokładność i strata po 20 epokach



### Dokładność i strata po 50 epokach

## Wizualizacja ekstraktora cech

Poniżej znajdują się wykresy t-SNE dla wyekstrahowanych cech ze zbioru treningowego i walidacyjnego:



# Podsumowanie

Porównanie modelu VGG-16 z przygotowanym przez nas modelem CNN wykazało, że VGG-16 osiągnął wyższą dokładność i charakteryzował się lepszym dopasowaniem, co sugeruje jego zdolność do generalizacji. Niemniej jednak, jesteśmy zadowoleni z naszej pracy, której efektem jest własny model również wykazujący zadowalającą stabilność i skuteczność, choć osiągnięte wyniki nie dorównały modelowi VGG-16. Mogliśmy także przekonać się jak ilość danych i ich różnorodność, struktura modelu wraz z jego cechami oraz ilość epok, przez które jest uczony wpływa na końcowy rezultat.

# Literatura i źródła

* Sandipan Dey, Python Image Processing Cookbook, 2020 Packt Publishing
* Tarek Amr, Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits, 2020 Packt Publishing
* Alberto Artasanchez, Prateek Joshi, Artificial Intelligence with Python Second Edition, 2020 Packt Publishing
* Antonio Gulli , Sujit Pal, Deep Learning with Keras, 2017 Packt Publishing
* <https://www.tensorflow.org/learn?hl=pl>
* <https://www.tensorflow.org/api_docs/python/tf/all_symbols>
* <https://keras.io/api/>
* <https://scikit-learn.org/stable/index.html>