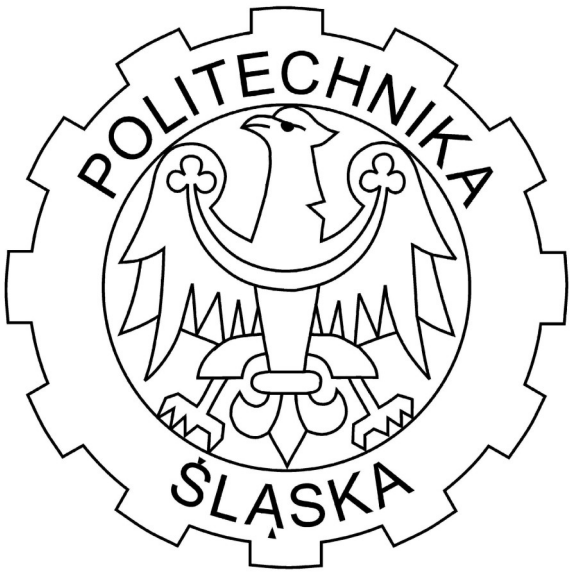
# Biologically Inspired Artificial Intelligence

# Raport z projektu semestralnego

**Temat projektu**

Rozpoznawanie charakterystycznych zawodów z obrazów




**Skład sekcji:** Dawid Musiał, Michał Rzepka
**Kierunek:** Informatyka [SSI]
**Grupa dziekańska:** ISMiP
**Semestr:** VI
**Adres e-mail:** dawimus184@student.polsl.pl lub michrze558@student.polsl.pl

# 1. Opis projektu

Celem tego projektu było stworzenie głębokiej sieci neuronowej, zdolnej do rozpoznawania zawodów spośród wybranych wcześniej kategorii za pomocą dostarczonych zdjęć. Główne zadanie polegało na napisaniu programu, który stworzy sieć neuronową, która wyuczy się rozpoznawania profesji osoby widniejącej na zdjęciu z dziesięciu możliwych klas.

# 2. Opis zewnętrzny aplikacji

Do uruchomienia programu należy pobrać i zainstalować kompilator do języka Python w wersji co najmniej 3.0 oraz za pomocą polecenia **`pip`** zainstalować biblioteki i ich zależności jakimi są: Keras, Matplotlib oraz Numpy. Kod źródłowy rozwiązania znajduje się w repozytorium dostępnym pod adresem: https://github.com/dmusial98/BIAI.git. Uruchomienie procesu tworzenia głębokiej sieci neuronowej i uczenie jej następuje po uruchomieniu pliku poprzez wpisanie w terminalu polecenia **`python3 main.py`** w folderze Release repozytorium. Trzeba zaznaczyć, że w repozytorium jest dostępne wiele folderów z różnymi wersjami uczenia sieci neuronowej. Największą skutecznością (wskaźnikiem prawidłowych przewidywań) odznacza się projekt znajdujący się w folderze v.6 ekstrakcja z augmentacją.

# 3. Opis wewnętrzny aplikacji

**Krótki opis kodu źródłowego:**

Na początku każdej z wersji projektów znajdują się ścieżki do katalogów ze zdjęciami służącymi do trenowania, testowania oraz walidacji (o tych procesach będzie wspomniane później) sieci. W każdym z tych katalogów znajduje się 10 podkatalogów z nazwami każdego z zawodów (**kucharz, lekarz, inżynier budowlany, rolnik, strażak, sędzia, mechanik samochodowy, pilot, policjant, kelner)** oraz zdjęcia jednej z wspomnianych wcześniej profesji. Każde ze zdjęć posiada wymiary 224x224 pikseli. Zbiór testowy zawiera po 900 zdjęć dla każdego zawodu (łącznie 9000 zdjęć). Zbiór testowy zawiera po 200 zdjęć (łącznie 2000 zdjęć testowych), a zbiór walidacyjny po 150 zdjęć (łącznie 1500 zdjęć).

Następnie w początkowych wersjach programu znajduje się kod odpowiedzialny za stworzenie nieskomplikowanej, stosunkowo „płytkiej” sieci neuronowej, składa się ona z warstw. Warstwą nazywamy podstawową strukturę danych sieci neuronowych. Jest ona modułem przetwarzania danych, który przyjmuje dane wejściowe z jednego tensora (lub kilku tensorów) i generuje dane w postaci tensora lub kilku tensorów (Tensor jest podstawową strukturą danych w systemach uczenia maszynowego, które zazwyczaj są reprezentowane przez liczby). Większość warstw jest charakteryzowanych przez stan – wagę. Wagę tworzy tensor lub kilka tensorów. Wartości zaś są ustalane przy użyciu algorytmu

stochastycznego spadku wzdłuż gradientu (o czym w dalszej części raportu). Nasza sieć składa się z konwolucyjnych warstw **Conv2D** służących do pracy z dwuwymiarowymi obrazami (czarno-białymi, bądź kolorowymi). Sieć zawiera również warstwy **MaxPooling2D** służące w skrócie głównie do zmniejszania rozdzielczości zdjęcia za pomocą okien ekstrakcji o wymiarach 2x2 przetwarzających mapy cech, okna te zwracają maksymalne wartości każdego z kanałów. Następnym typem warstw jest warstwa **Flatten**, która spłaszcza wielowymiarowe próbki danych do postaci jednowymiarowej. Ostatnim typem warstwy jest warstwa **Dense**, tak zwana warstwa gęsta, która służy głównie do uczenia się sieci poprzez wagi w swojej strukturze, wagi te są właśnie wiedzą danej sieci. Sieci gęste **Dense** najczęściej są górnymi warstwami sieci. Na wyjściu każdej z sieci **Conv2D** oraz **MaxPooling2D** pojawia się trójwymiarowy tensor o kształcie: wysokość, szerokość, kanały (3 wymiary). Wraz z zagłębianiem się w sieć wysokość i szerokość mają tendencję do przyjmowania mniejszych wartości. Liczbę kanałów określa pierwszy argument przekazany do warstw **Conv2D** (32 lub 64 kanały). Po spłaszczeniu ostatniego wielowymiarowego tensora o kształcie 12,12,128 (patrz output wersji pierwszej) do tensora jednowymiarowego (płaskiego), dane przekazywane są do pierwszej warstwy gęstej sieci o rozmiarze próbek 512. Ostatnia gęsta warstwa przyjmuje próbki, które później klasyfikuje do jednej z dziesięciu klas zdjęć, jej funkcją aktywacji jest funkcja **softmax**, która jest pomocna przy klasyfikacji obiektów do jednej z wielu klas (np. zdjęć). Warstwy konwolucyjne używają funkcji aktywacji **relu**, która jest typem funkcji nieliniowej pomagająca jeszcze wydajniej wykorzystać możliwości warstw **Conv2D.**

W późniejszych wersjach programu zastosowano również warstwę **Dropout**, która ma za zadanie odrzucać losowo różne podzbiory neuronów podczas przetwarzania każdego przykładu. Warstwa **Dropout** służy zapobieganiu nadmiernemu dopasowaniu do danych. Nadmierne dopasowanie to innymi słowy sytuacja, gdy sieć na zbiorze treningowym wyuczyła się niepotrzebnych cech. Z nadmiernym dopasowaniem mamy do czynienia, gdy sieć rozpoznaje zdjęcia ze zbioru treningowego ze skutecznością sięgającą około 98 – 99,9%, zaś ze zbioru walidacyjnego i testowego już ze znacznie mniejszą. We wszystkich późniejszych wersjach programu znajduje się jedna warstwa **Dropout** z współczynnikiem porzucania 0,5, co oznacza, że sieć porzuca losowo połowę osiągniętych wyników. Następnie następuje wybranie funkcji straty (którą opiszemy w dalszej części) jaką jest entropia krzyżowa kwalifikująca dane do jednej z klas. Jako metrykę do monitorowania wyuczenia sieci przyjęto skuteczność przewidywania klasy, do której należy dany obraz.

Kolejnym krokiem jest przeskalowanie każdej z wartości z zakresu 0 - 255 kanałów RGB na liczby zmiennoprzecinkowe z zakresu [0; 1]. W dalszej części programu następuje zdefiniowanie treningowego, walidacyjnego i testowego generatora, które będą dostarczać dane w określony sposób. W każdym z generatorów określa się ścieżkę do danych, wymiar (rozdzielczość) próbek danych, ilość próbek podawanych w pojedynczym wsadzie oraz model klasyfikacji, w naszym wypadku będzie to zawsze klasyfikacja związana z wieloma kategoriami: **„categorical”**.

Bardzo ważną częścią programu jest dopasowanie modelu do danych przy użyciu generatora za pomocą metody **fit_generator**. Metoda ta oczekuje zdefiniowania

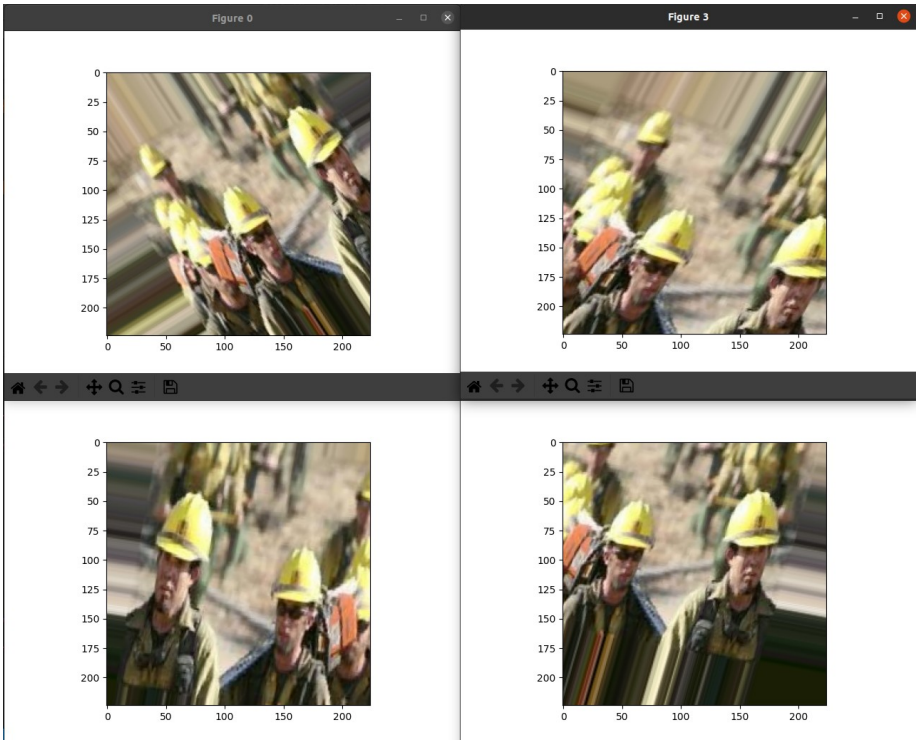
w pierwszym argumencie generatora, który w nieskończoność będzie zwracał wsady danych wejściowych i ich etykiet. Dane są generowane w nieskończoność, a więc model Keras musi wiedzieć, ile próbek ma pobrać z generatora przed zakończeniem epoki. Służy do tego argument **steps_per_epoch**. Następnie podajemy ilość epok trenowania modelu. W metodzie **fit_generator** można również przekazać argument **validation_data**, który przekazuje w nieskończoność próbki danych walidacyjnych, dlatego kolejnym argumentem jest wielkość próbki wsadu dla jednej epoki.

W większości wersji programu ostatnią jego częścią jest wygenerowanie wykresów obrazujących proces uczenia modelu sieci. Pierwszy z nich obrazuje dokładność trenowania i walidacji w poszczególnych epokach, zaś drugi stratę trenowania i walidacji,

W późniejszych wersjach projektu zastosowano technikę augmentacji danych, której głównym zadaniem jest walka z nadmiernym dopasowaniem do zbioru testowego poprzez losowe przekształcanie obrazów wejściowych, przykładowy kod augmentacji wraz z objaśnieniem znajduje się poniżej:

```
train_datagen = ImagaeDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

Pierwszym argumentem jest przeskalowanie każdej z wartości kanałów RGB do liczby zmiennoprzecinkowej z przedziału [0; 1], drugim **rotation_range** – zakres kątów, o które zostanie wykonany losowy obrót obrazów, trzecim i czwartym **width_shift**, **height_shift**, które określają ułamki całkowitej szerokości i wysokości obrazów, zakresy te wskazują ramy, w obrębie których przeprowadza się losowe pionowe i poziome przekształcenia obrazów. Parametr **shear_range** określa zakres losowego przycinania obrazu. Parametr **zoom_range** określa zakres losowego przybliżania fragmentów obrazów. Operacja **horizontal_flip** polega na losowym odbiciu połowy obrazu w płaszczyźnie poziomej – z przekształcenia tego warto korzystać wtedy, gdy nie ma założeń o horyzontalnej asymetrii obrazu (np. w przypadku prawdziwych zdjęć). Tryb **fill_mode** jest strategią wypełniania nowo utworzonych pikseli, które mogą powstać w wyniku obrotu lub przesunięcia. Przykłady przekształconych obrazów można zobaczyć poniżej:




W tym miejscu należy zwrócić uwagę na fakt, że po wykorzystaniu augmentacji danych oraz warstwy **Dropout** z współczynnikiem porzucenia 0,5, dokładność trenowania i walidacji osiąga niemal taką samą wartość, co świadczy o braku zjawiska przeuczenia sieci (nadmiernego dopasowania).

Ostatnim większym wykorzystanym narzędziem w procesie tworzenia sieci jest ekstrakcja cech sieci. Ekstrakcja cech polega na korzystaniu z reprezentacji wyuczonej przez sieć wcześniej w celu dokonania ekstrakcji interesujących nas cech z nowych próbek. Cechy te są następnie przepuszczane przez nowy klasyfikator trenowany od podstaw. W przypadku konwolucyjnej sieci neuronowej ekstrakcja cech polega na przyjrzeniu się konwolucyjnej bazie wcześniej wytrenowanej sieci, przepuszczeniu przez nią nowych danych i wytrenowaniu nowego klasyfikatora na bazie wyjścia tej sieci. W bazie wcześniej wytrenowanej sieci znajdują się ogólne koncepcje pracy ze zdjęciami i ich przetwarzania, takie jak rozpoznawanie krawędzi itp. Klasyfikator jest złożony z warstw, które służą głównie do klasyfikacji obrazów do jednej z klas, niekoniecznie tych, których używamy w naszych projektach. W naszym projekcie użyliśmy konwolucyjnej bazy sieci VGG16. Kod odpowiedzialny za dołączenie bazy tej sieci jest zamieszczony poniżej:

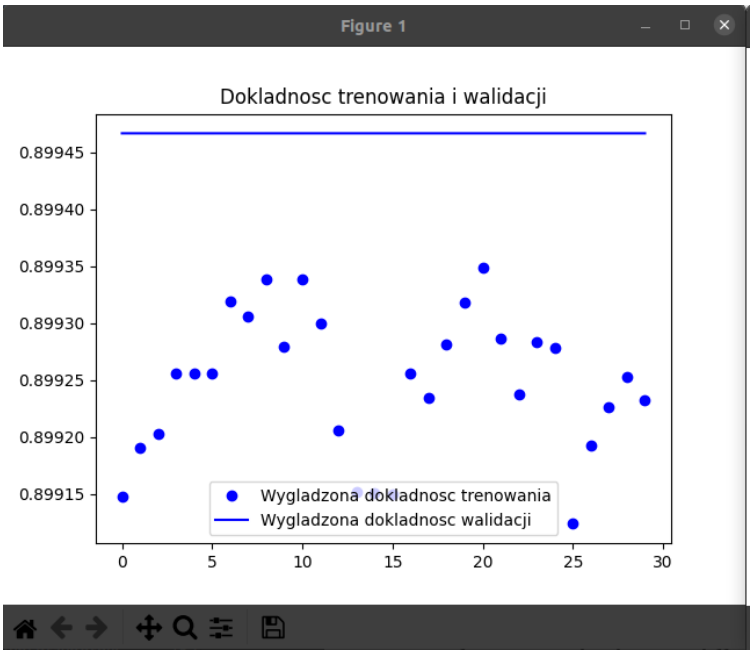
```
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(224,224,3))
```

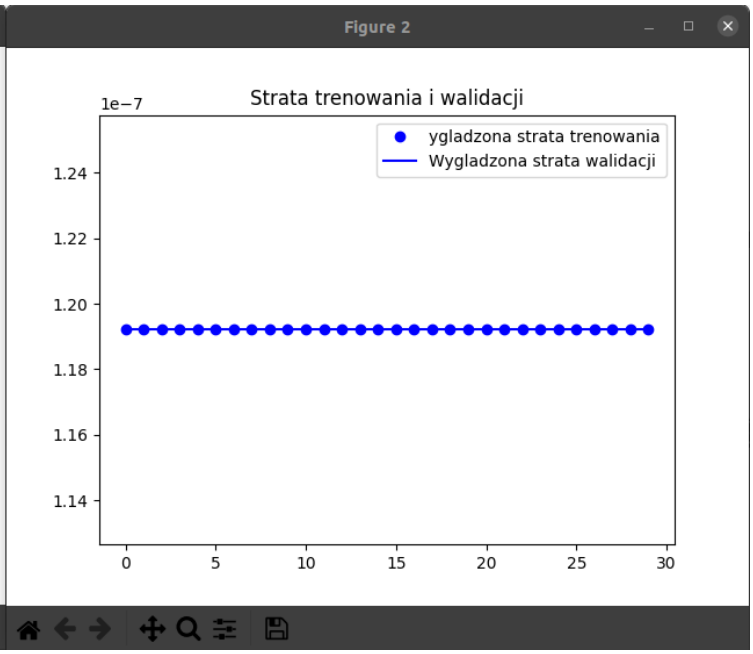
Do konstruktora modelu przekazano trzy argumenty:
- **weights** – określa punkt kontrolny wag, z którego inicowany jest model
- **include_top** – określa czy do górnej części sieci dołączony zostanie gęsto połączony klasyfikator. (W naszym wypadku świadomie odrzuciliśmy go).
- **input_shape** – określa kształt tensorów obrazów, które będą kierowane do wejścia sieci.

Następnie wykonano szybką ekstrakcję cech z pominięciem augmentacji danych, proces ten trwał bardzo krótko, bo jedynie niecałe 3 minuty w przypadku 100 epok.

Ostateczna wersja programu (z największą skutecznością w rozpoznawaniu i klasyfikowaniu obrazów) korzysta z ekstrakcji cech z augmentacją danych. Proces ten uruchamiany tylko na procesorze CPU pracował przez 60 godzin z bardzo dobrym rezultatem, wprawdzie można było go odciążyć przez przekazanie części obliczeń do procesora graficznego, jednak podczas realizacji projektu występowały problemy z zainstalowaniem odpowiednich sterowników do karty graficznej, dlatego poprzestano na użyciu procesora CPU. Nasz model sieci zawierał bazę konwolucyjną **`conv_base`** oraz warstwę spłaszczającą dane **`Flatten`** i dwie warstwy głębokie **`Dense`**. Podczas trenowania sieci z tą techniką należało również zamrozić bazę konwolucyjną w celu zapobiegania aktualizowaniu wag określonych warstw. Ostatnią częścią finalnej wersji było dostrajanie sieci konwolucyjnej tzn. po wytrenowaniu własnej części sieci (klasyfikatora) zaczęto odmrażać niektóre z warstw bazy konwolucyjnej. Proces ten nieznacznie poprawił wyniki osiągnięte przez sieć.

Podsumowując pierwotna wersja sieci osiągała skuteczność na poziomie około 60%. Wersja z augmentacją danych potrafiła osiągnąć do 73% skuteczności. Wersja z ekstrakcją bez augmentacji danych dała wyniki na poziomie 76%, zaś ekstrakcja cech z augmentacją danych i dostrajaniem pojedynczych warstw osiągnęła wynik na poziomie 90%. Naszym zdaniem jest to satysfakcjonujący rezultat. Poniżej zamieszczamy wykresy z ostatniej próby trenowania sieci.



# 4. Algorytm

Główny algorytm odpowiedzialny za naukę sieci polega na obraniu trzech ważnych czynników określających pracę sieci są nimi:
- funkcja straty – funkcja ta definiuje sposób pomiaru wydajności sieci podczas przetwarzania treningowego zbioru danych, a więc pozwala na dostrajanie parametrów sieci we właściwym kierunku;
- optymalizator – mechanizm dostrajania sieci na podstawie danych zwracanych przez funkcję straty;
- metryki monitorowane podczas trenowania i testowania – tutaj interesuje nas jedynie dokładność (część obrazów, która została właściwie sklasyfikowana).

Sam proces trenowania polega na wykonywania liniowych obliczeń na danych wchodzących do warstwy sieci oraz tych zapamiętanych w warstwie. Pewne parametry są zapamiętywane w sieci podczas procesu uczenia, nazywane są także potocznie wagami. Początkowo te matryce wag są wypełniane małymi losowymi wartościami. Następnie wagi są dostrajane na podstawie sygnału zwrotnego. Proces ten właśnie nazywany trenowaniem, jest on najważniejszym elementem uczenia maszynowego. Przeprowadzany jest on za pomocą pętli trenowania, która składa się z następujących operacji, które są powtarzane, aż do uzyskania satysfakcjonujących wyników:

1. Wygenerowanie wsadu składającego się z próbek treningowych **`x`** i odpowiadających im wartości docelowych **`y`**.
2. Uruchomienie sieci na próbkach **`x`** w celu uzyskania przewidywanych wartości **`y_pred`**.
3. Obliczenie straty sieci na próbce danych i zmierzenie różnicy między wartościami **`y_pred`** i **`y`**.
4. Obliczenie gradientu straty w odniesieniu do parametrów sieci.
5. Przesunięcie wartości w kierunku przeciwnym do gradientu, co powoduje małe zredukowanie straty uzyskanej na bieżącej próbce danych treningowych.

Punkt numer 4 i 5 zmierza do operacji wygenerowania funkcji, która ma za zadanie podawać informację zwrotną w procesie trenowania sieci, mówiącej o tym w jakim stadium uczenia się jest obecnie sieć i ile wynosi skuteczność przewidywania. Proces ten przebiega poprzez stochastyczny spadek wzdłuż gradientu. W skrócie opiera się on na wykorzystaniu pojęcia pochodnej i umiejętności jej wyznaczenia w procesie trenowania sieci. W metodzie tej jednak nie wykorzystujemy różniczkowalności funkcji jednej zmiennej, a wielu zmiennych, a w zasadzie różniczkowalności tensora. Wynik takiego różniczkowania tensora nazywamy gradientem. Sam tensor będący wynikiem różniczkowania pewnego tensora może być interpretowany jako pochodna, a więc opisanie krzywizny funkcji f(W) w punkcie $W_0$. Dysponując funkcją różniczkowalną teoretycznie możemy znaleźć jej minimum, ponieważ wiemy, że minimum funkcji jest punktem, w którym jej pochodna wynosi 0. W związku z tym wystarczy znaleźć wszystkie punkty, w których pochodna funkcji uzyskuje wartość 0, a następnie sprawdzić wartość funkcji w tych punktach i wybrać najniższą z nich. Jednak rozwiązanie takiego równania ze względu na bardzo dużą ilość zmiennych jest bardzo

pracochłonne, dlatego lepiej wykorzystać metodę, która korzysta z powolnego przesuwania wartości argumentu w kierunku przeciwnym do krzywizny (gradientu). Jeżeli zaktualizujemy wagi w kierunku przeciwnym do do gradientu, strata będzie zmniejszana przy każdym wykonaniu algorytmu. Przy wykonywaniu tego algorytmu trzeba jeszcze uwzględnić jeden bardzo ważny aspekt – dobranie odpowiedniego skoku wzdłuż każdej z osi, ponieważ jak wiadomo każde minimum lokalne ma w swoim otoczeniu po obu stronach wartości większe od siebie, dlatego potrzebna jest wartość pozwalająca „przeskakiwać" pomiędzy wieloma ekstremami w celu znalezienia minimum funkcji w interesującym nas przedziale. Innymi słowy złe dobranie tego parametru może skutkować zatrzymaniem się funkcji obliczającej straty wokół jednego minimum lokalnego, pomijając pozostałe.

W naszym projekcie poruszono kwestię trenowania, walidacji i testowania sieci neuronowej, każdy z tych procesów ma inne zadanie. Generator testowy ma za zadanie dostarczać wsady danych do uczenia się sieci poszczególnych cech dla danego zawodu na zdjęciu. Proces walidacji polega na dostarczaniu małych ilości danych po zakończeniu każdej z epok w celu monitorowania dokładności modelu. Jednocześnie monitorujemy funkcję straty i dokładność modelu przy przetwarzaniu próbek ze zbioru walidacyjnego. Warto tutaj zaznaczyć, że każde ze zdjęć występuje tylko w jednej z trzech grup (treningowa, walidacyjna i testowa). Zbiór testowy służy do testowania wyników wyuczenia sieci poprzez dostarczanie nowych jeszcze nigdy nie spotkanych przez sieć zdjęć i sprawdzenie jej skuteczności w przewidywaniu.

W tym paragrafie chcielibyśmy opisać jeszcze pokrótce działanie konwolucyjnych sieci neuronowych. Podstawową różnicą pomiędzy warstwą gęstych połączeń, a siecią konwolucyjną jest to, że warstwy **Dense** uczą się cech parametrów globalnych w swoich wejściowych przestrzeniach (głównie są to wzorce związane związane ze wszystkimi pikselami), a warstwy konwolucyjne uczą się lokalnych wzorców – w przypadku obrazów wzorce są znajdowane w małych dwuwymiarowych oknach danych wejściowych (zwykle o wymiarach od 2x2 do 5x5). Dzięki temu wzorce rozpoznawane przez sieć są niezależne od przesunięcia. Sieć konwolucyjna po rozpoznaniu określonego wzoru w prawym górnym rogu obrazu może rozpoznać go np. w lewym górnym rogu. Do tego sieci konwoluzyjne mogą uczyć się przestrzennej hierarchii. Pierwsza warstwa sieci konwolucyjnej uczy się małych lokalnych wzorców, takich jak krawędzie, druga warstwa tej sieci będzie uczyła się większych struktur składających się z elementów rozpoznanych przez pierwszą warstwę itd. Sieci konwolucyjne działają na trójwymiarowych tensorach określanych mianem map cech, zawierających dwie przestrzenne osie definiujące wysokość i szerokość. Trzecią osią jest oś głębi, nazywana również osią kanałów. W przypadku obrazu RGB oś głębi ma trzy wymiary. Sieci konwolucyjne dokonują ekstrakcji łat z cech wejściowych i przeprowadzają tę samą operację na wszystkich łatach w celu utworzenia utworzenia wyjściowej mapy cech. Mapa ta jest nadal trójwymiarowym tensorem, ale jej głębia może mieć dowolną wartość, ponieważ wyjściowa wartość głębi jest parametrem warstwy, a różne kanały osi głębi nie określają już nasycenia poszczególnych kolorów składowych. Teraz są one filtrami. Filtry kodują określone cechy danych wejściowych: na wysokim poziomie pojedynczy filtr może np. kodować obecność twarzy w obrazie wejściowym. Konwolucja polega na przesuwaniu okien o wymiarach 3x3 lub 5x5

po trójwymiarowej mapie cech. Okno jest zatrzymywane w każdym miejscu, w którym da się je umieścić i dokonywana jest ekstrakcja trójwymiarowej łaty otaczających je cech. Każda taka łata jest następnie przekształcana w celu uzyskania jednowymiarowego wektora. Wszystkie te wektory są następnie przestrzennie przebudowywane w celu utworzenia trójwymiarowej mapy wyjściowej.

# 5. Opis działania

Program po uruchomieniu tworzy głęboką sieć neuronową, włącznie z własnym klasyfikatorem, inicjalizuje generator odpowiedzialny za przeskalowanie wartości każdego z kanałów do liczb zmiennoprzecinkowych z zakresu [0;1], następnie rozpoczyna proces augmentacji danych według określonych argumentów. Dalej następuje inicjalizacja generatorów treningowego, walidacyjnego i testowego. Później sieć poddawana jest procesowi trenowania z walidacją wyników po zakończeniu każdej z epok. Po zakończeniu trenowania wkracza proces dostrajania sieci poprzez odmrażanie pewnych warstw głębokiej już wyuczonej sieci. Ostatnim etapem programu jest zwizualizowanie wygładzonych wykresów dokładności trenowania i dokładności walidacji oraz straty trenowania i straty walidacji.

# 6. Opis wykorzystywanych bibliotek

Keras - biblioteka będąca ramą projektową uczenia głębokiego języka Python, która umożliwia wygodne definiowanie i trenowanie dowolnych modeli uczenia głębokiego. Głównymi cechami tej biblioteki są:

- możliwość bezproblemowego uruchamiania tego samego kodu na procesorach CPU i GPU
- przyjazny interfejs programistyczny ułatwiający szybkie prototypowanie modeli uczenia głębokiego
- wbudowana obsługa sieci konwolucyjnych służących do przetwarzania obrazu, sieci rekurencyjnych służących do przetwarzania danych sekwencyjnych i sieci będących połączeniem obu tych rozwiązań
- obsługa sieci o dowolnych architekturach: modeli obsługujących wiele wejść i wyjść; możliwość współdzielenia warstw i modeli. Dzięki tym możliwościom pakiet Keras nadaje się do budowania praktycznie dowolnych modeli uczenia głębokiego - od generatywnych sieci z przeciwnikiem do neuronowych maszyn Turinga.

Z biblioteki Keras korzystają firmy takie jak Google, Netflix, Uber czy CERN. Sam Keras zapewnia wysokopoziomowe bloki, z których powstają modele uczenia głębokiego. Biblioteka nie definiuje operacji niskopoziomowych takich jak operacje na tensorach.

Numpy - biblioteka używana do wszelkich obliczeń (np na tensorach). Jej interfejs jest również bardzo intuicyjny i znacznie ułatwia pracę z tensorami podczas projektowania sieci głębokich.
Matplotlib - biblioteka służąca do tworzenia wykresów w języku Python. w zasadzie napisanie kilkunastu linii kodu owocuje bardzo czytelnym i przejrzystym wykresem.

# 7. Wnioski

Sieci konwolucyjne okazują się bardzo dobrą strukturą do pracy z rozpoznawaniem obrazów i ich klasyfikacji. Ich sposób działania polegający na szukaniu wzorców takich jak krawędzie (w niższych warstwach), oraz większych elementów (np. człowieka) w wyższych warstwach jest świetną taktyką do rozwiązywania tego typu problemów. Finalna skuteczność sięgała 90% skuteczności co oceniamy jako satysfakcjonujący wynik. W tym miejscu trzeba zaznaczyć, że nasz zbiór danych nie był bardzo duży, dlatego gdybyśmy dysponowali zbiorem danych np. 10 razy większym przypuszczamy, że wyniki mogłyby być jeszcze lepsze. Na początku implementacji borykaliśmy się z problemem nadmiernego dopasowania, jednak zastosowanie warstwy porzucającej (Dropout) i techniki augmentacji danych w zasadzie całkowicie zniwelował ten efekt. Wykorzystanie wcześniej wytrenowanej sieci konwolucyjnej VGG16 pomogło również znacznie poprawić wynik.

Podczas implementowania tego projektu mogliśmy oboje po raz pierwszy zetknąć się ze sztuczną inteligencją i procesem tworzenia sieci głębokich. W naszej pracy bardzo pomocna przydała się książka Francois'a Chollet'a pt. "Deep Learning. Praca z językiem Python i biblioteką Keras.". Zawiera ona teoretyczny opis i sposób działania komputerowo projektowanych neuronowych sieci głębokich. Po wstępie teoretycznym jest również wiele dobrze opisanych przykładów pokazujących jak należy pracować z biblioteką Keras.

Sam proces wytwarzania programu był dla nas bardzo przyjemny, stosunkowo intuicyjny. Oboje zgodnie stwierdziliśmy, że sztuczna inteligencja niesie ze sobą duży potencjał, który będziemy chcieli wykorzystać we własnych projektach w przyszłości.