|  |
| --- |
| agh_nzw_s_pl_1w_wbr_rgb_150ppi |
| **WYDZIAŁ INFORMATYKI, ELEKTRONIKI i TELEKOMUNIKACJI**  KATEDRA TELEKOMUNIKACJI |
| **PRACA DYPLOMOWA INŻYNIERSKA** |
| **Implementacja algorytmu wykrywania ognia na obrazie z kamery wizyjnej.** |
| *Implementation of the fire detection algorithm based on a video camera image.* |
| Autor: **Karol Chomiuk**  Kierunek studiów: Elektronika i Telekomunikacja  Typ studiów: Stacjonarne  Opiekun pracy: dr inż. Andrzej Matiolański |
| Kraków, 2019 |

OŚWIADCZENIE STUDENTA

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz.U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym (tj. Dz.U. z 2018 r. poz. 1668, z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz.U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

(*czytelny podpis studenta*)

Spis treści

[1. Wstęp 3](#_Toc33093050)

[2. Opis wykorzystywanych narzędzi 4](#_Toc33093051)

[2.1. Język Python 4](#_Toc33093052)

[2.2. Biblioteki Keras oraz Tensorflow 4](#_Toc33093053)

[2.3. Biblioteki 5](#_Toc33093054)

[3. Pojęcie uczenia maszynowego 6](#_Toc33093055)

[4. Głębokie sieci neuronowe (CNN) 8](#_Toc33093056)

[4.1.1. Pojęcia związane z uczeniem maszynowym 9](#_Toc33093057)

[5. Projektowanie 11](#_Toc33093058)

[5.1. Import bazy danych 13](#_Toc33093059)

[5.2. Przetwarzanie wstępne zdjęć oraz podział 14](#_Toc33093060)

[5.3. Opracowanie modelu 14](#_Toc33093061)

[5.4. Trenowanie modelu 15](#_Toc33093062)

[5.4.1. Tworzenie modelu 15](#_Toc33093063)

[5.4.2. Learning rate 17](#_Toc33093064)

[5.4.3. Trenowanie modelu 19](#_Toc33093065)

[5.5. Predykcja, dokładność 24](#_Toc33093066)

[5.5.1. Analiza całych obrazów 25](#_Toc33093067)

[5.5.2. Analiza pojedynczych fragmentów obrazów 26](#_Toc33093068)

[6. Podsumowanie 30](#_Toc33093069)

[7. Bibliografia 31](#_Toc33093070)

# Wstęp

Celem pracy jest praktyczne przedstawienie działania algorytmu wykrywającego ogień na podstawie obrazu z kamery wizyjnej. Zastosowano w tym celu nadzorowane uczenie maszynowe. Poprzez zaimportowanie zbioru danych liczącego ponad 1000 zdjęć, zawierającego obrazy ognia w różnym środowisku oraz ponad 3000 w sytuacjach, gdzie nie istnieje ryzyko pożaru, program ma za zadanie wykryć pożądane zjawisko i poinformować o nim użytkownika. Projekt został wykonany przy użyciu języka Python oraz skorzystano z powszechnie dostępnej/ogólnodostępnej (ang. open-source’owej) biblioteki Keras.

Realizacja niniejszej pracy może posłużyć jako dodatkowe zabezpieczenie w przypadku pożarów. Stosowane ówcześnie zabezpieczenie w postaci czujników dymu działa bez zarzutów, jednak na otwartej przestrzeni traci on swoją funkcjonalność. Zastosowanie przedstawionego w tej pracy rozwiązania, pozwoliłoby na wczesne wykrycie ognia w przestrzeniach miejskich, co zmniejszyłoby czas pomiędzy wystąpieniem zdarzenia, a zawiadomieniem odpowiednich służb. w  konsekwencji czego, zagrożenie życia oraz zaistniałe szkody będą zminimalizowane. Zbiór danych (ang. Data set) zawierał będzie obrazy ognia w kontrolowanych i niekontrolowanych warunkach, a także zdjęcia w których żadne zagrożenie nie występuje.

Rozdział pierwszy jest wprowadzeniem do poruszanego tematu. Rozdział drugi omawia wykorzystane w pracy narzędzia takie jak środowisko oraz wykorzystane biblioteki. w  trzecim rozdziale zawarta została definicja pojęcia uczenia maszynowego (ang. Machine Learning) i dokładniejsza analiza teoretycznego działania algorytmu. Kolejny rozdział – nr 4 to teoretyczny opis funkcjonowania sieci neuronowych. Projekt aplikacji został zawarty w rozdziale piątym, w którym w sposób praktyczny rozwiązywany jest badany problem. Zawiera on szczegóły procesu tworzenia odpowiedniej bazy danych oraz uczenia modelu. Rozdział szósty stanowi podsumowanie realizowanego projektu i wnioski płynące z zastosowanego rozwiązania. Niniejsza praca oparta została na naukowej literaturze opisanej w bibliografii stanowiącej ostatni rozdział pracy.

# Opis wykorzystywanych narzędzi

Niniejsza praca dyplomowa dzieli się na część teoretyczną oraz praktyczną. w pierwszej z nich przedstawiony zostanie ogólny problem badawczy oraz sposób na jego realizację. Zaś druga część, stanowi programistyczne rozwiązanie wraz z dogłębną analizą przeprowadzanych badań.

## Język Python

Pierwszą decyzją jest wybór odpowiedniego języka programowania. w  przypadku uczenia maszynowego najlepsza decyzją jest język Python[1]. Oferuje on mnogość bibliotek przeznaczonych do tego zagadnienia m.in. Keras[2], TensorFlow[3], Pandas[4] czy Scikit-learn[5]. Jest to bardzo łatwy do zrozumienia, intuicyjny i czytelny język. Posiada on bardzo rozbudowaną społeczność przez co jest stale udoskonalany. Czytelność kodu jest kolejną zaletą przemawiającą za tym wyborem. Prostota składni pozwala skupić się na innych problemach związanych z uczeniem maszynowym niż debugowanie kodu.

Innym językiem do zastosowań uczenia maszynowego może być również Java. Jest ona szybsza w porównaniu do języka Python. Używana jest przede wszystkim w środowisku gier i mobilnych aplikacji. w celach dogłębniejszej analizy i statystyki można również wykorzystać język R, używany przede wszystkim w branży Data Science, czyli dziedziną zajmującą się wyciąganiem wniosków z dostarczonych danych. [6]

## Biblioteki Keras oraz Tensorflow

Kolejną decyzją, jaką należy podjąć przy rozpoczęciu pracy z uczeniem maszynowym jest wybór odpowiednich bibliotek (ang. library) [7] Są to pliki obiektowe, z których zalet możemy korzystać z poziomu kodu źródłowego bez konieczności ich ponownej implementacji. Python oferuje mnogość wbudowanych bibliotek np. Glob z której funkcji możemy korzystać dzięki uprzedniemu zaimportowaniu. Jednak na potrzeby pracy zostały dołączone również biblioteki zewnętrzne, co zostało opisane w dalszej części pracy. Na potrzeby niniejszej pracy dyplomowej korzystał z open source’owej biblioteki- Keras, przeznaczonej do pracy z głębokimi sieciami neuronowymi[8]. Została ona zaprojektowana w oparciu o język programowania Python jako API czyli interfejs programowania aplikacji, a nie jako samodzielna platforma i jest kompatybilna z takimi platformami jak TensorFlow (TF), CNTK[9] lub Theano[10]. Keras bazuje na bibliotece TensorFlow jednak jest on bardziej przyjazny użytkownikowi [11].

Nasunąć się może zatem pytanie dlaczego wykorzystywane są poszczególne biblioteki i jakie są między nimi różnice. Keras jest następcą TensorFlow i korzysta z jego zalet, do których można zaliczyć modułowość, która w przyjazny sposób pozwala na łączenie kolejnych wymagań użytkownika. [12]. w  łatwy i szybki sposób przy jego pomocy można zaimplementować pierwszy model. Zaletą TF jest fakt, że jest to środowisko dużo bardziej elastyczne i pozwalające użytkownikowi dodawać własne biblioteki. Warto dodać, że TF oferuje więcej zaawansowanych operacji jak debugowanie oraz obsługę wątków i kolejek[13]. Keras jest narzędziem dużo bardziej intuicyjnym i prostszym. Korzysta on z dogodności oferowanych przez TensorFlow, jednocześnie, poprzez swoją łatwość w nauczeniu się, pozwalają użytkownikowi być bardziej produktywnym. Ponadto posiada bardzo szeroką społeczność oraz rozbudowane repozytorium co może być pomocne przy szukaniu rozwiązań do napotkanych problemów.

Do niniejszej pracy wykorzystywał będę bibliotekę Keras. Zakres jej działań jest wystarczający do moich potrzeb. Jako, że jest ona integralną częścią TensorFlow, zawsze pozostaje możliwość ulepszenia działania programu o jego zalety.

## Biblioteki

Oprócz biblioteki przeznaczonej do celów uczenia maszynowego, wykorzystane zostały inne biblioteki, zwłaszcza do przetwarzania oraz obrabiania obrazów. w  tym celu wykorzystywana została biblioteka OpenCV[14]*.* Przy jej pomocy stworzony został skrypt do obróbki zdjęć i poszerzenia bazy danych oraz zostały zaimportowane obrazy stanowiące bazę danych do nauki modelu. Znalazła ona ponadto zastosowanie przy etapie predykcji, gdzie pozwoliła na odczytywanie filmów oraz uwidocznienie miejsca wystąpienia ognia. Wybrałem tą bibliotekę, ponieważ oferowała ona wszystkie potrzebne funkcję do przetwarzania obrazów.

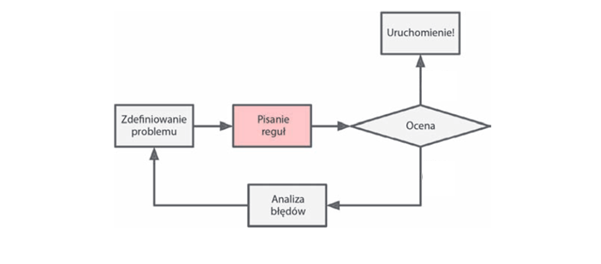
Ponadto wykorzystana została biblioteka NumPy[15] do pracy na macierzach oraz wbudowany moduł Glob w celu odczytywania nazw ścieżek plików.

W celu instalacji bibliotek, które nie należą do standardowych oferowanych przez Python, wykorzystany został menadżer pakietów PIP w wersji 3.7.

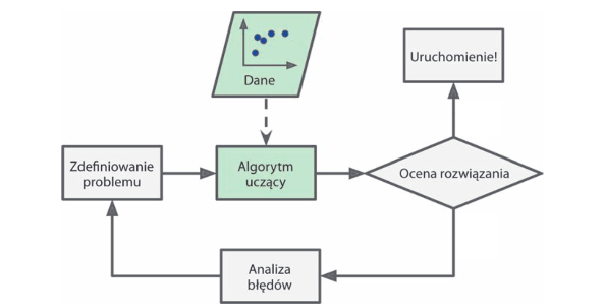
# Pojęcie uczenia maszynowego

Pojęcie uczenia maszynowego jest ściśle związane ze sztuczną inteligencją. Opiera się ono na pojęciach z dziedziny informatyki i statystyki. Jako pierwszy definicję przedstawił Arthur Samuel w 1959 roku określając *„Uczenie Maszynowe jako dziedzinę nauki dającą komputerom możliwość uczenia się bez konieczności jawnego programowania”*[12]*.* Oznacza to, że komputer wraz z pozyskiwanym doświadczeniem w postaci danych powinien usprawniać działanie programów poprzez zwiększanie ich wydajności, skuteczności oraz efektywności. Ogranicza to rolę człowieka w ciągłym udoskonalaniu kodu źródłowego. w przypadku zaimplementowanego uprzednio kodu korzystającego z zalet machine learning, usprawnianie działania polegać będzie na dostarczeniu nowych danych uczących, zaś w podejściu manualnym konieczna będzie ręczna zmiana napisanego wcześniej kodu.

Najłatwiej to zjawisko można opisać na przykładzie algorytmu wykorzystywanego do wykrywania spamu w wiadomościach E-Mail [17] Tak jak w przypadku każdego programu opartego ma metodyce Machine Learning, potrzebujemy odpowiedniej bazy danych różnych wiadomości, w których jednoznacznie określone jest, które z nich są wiadomościami niepożądanymi, aby wytrenować model[16]. Na ich przykładzie, komputer ma za zadanie nauczyć się, co zawierają podane wiadomości i przewidzieć, gdzie kolejne przychodzące maile mają trafić[17]. Im większa baza danych tym prawdopodobieństwo powodzenia operacji jest większe. Zbyt mała liczba danych może skutkować słabym przybliżeniem, ponieważ jest większa szansa na trafienie na przypadek, który będzie dla maszyny ciężki do zdiagnozowania. w przypadku dostarczenia dużej ilości maili zawierających niepożądane informacje, algorytm może reagować nie tylko na adresatów, ale na treść oraz zawartość wiadomości oraz dostosowywać metodykę działania nie tylko dla wybiórczych słow, ale również dla całych sekwencji. Nasunąć może się pytanie, dlaczego zatem nie napisać tego algorytmu w sposób tradycyjny, skoro i tak w początkowej fazie projektu potrzebna jest ingerencja człowieka. Kod nie korzystający z algorytmu maszynowego miałby poważną wadę – musiałby być stale aktualizowany pod kątem dopisywania kolejnych reguł filtrowania, a nadawcy niepożądanych wiadomości mogliby w łatwy sposób modyfikować je, w sposób który omijałby zabezpieczenie antyspamowe. z pomocą przychodzi Uczenie Maszynowe, które pozwala na stałą poprawę działania programu. Wystarczy jedynie dodać do bazy danych wiadomość, która jest tzw. spamem, zamiast dopisywania kolejnych reguł filtrowania wiadomości. Na podobnej zasadzie działania będzie opierał się program opisany w tej pracy dyplomowej. Aby pozwolić komputerowi na samodzielne przewidywanie zdarzeń, zostanie on nauczony, przy pomocy odpowiednio obszernej, etykietowanej bazie danych, jakie zjawiska powinny zostać wykrywane. Na rysunku 1. przestawione zostało tradycyjne podejście, a rysunek 2. obrazuje opracowanie rozwiązania korzystającego z zalet uczenia maszynowego.



Rysunek 1. Tradycyjne podejście[12]

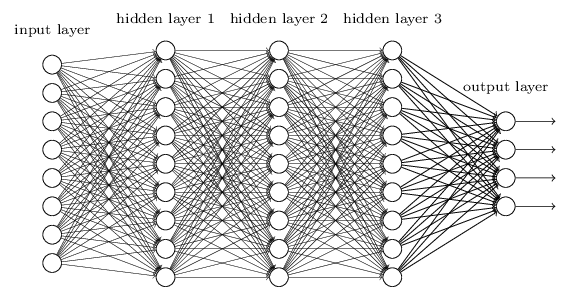


Rysunek 2. Podejście wykorzystujące uczenie maszynowe[12]

# Głębokie sieci neuronowe (CNN)

Do prawidłowego funkcjonowania programu nie wystarczy jedynie dobrze dobrana baza obrazów. Kolejnym równie ważnym krokiem jest stworzenie odpowiedniego modelu. Na potrzeby niniejszej pracy dyplomowej wykorzystywać będziemy CNN- Convolutional Neural Network czyli klasa głębokich sieci neuronowych[18]. Czym zatem są sieci neuronowe[19]?

Podstawowa wyżej wspomniana sieć składa się z warstwy wejściowej i wyjściowej oraz warstwy ukrytej pomiędzy nimi, zaś różnica między nią a głęboką polega na tym że warstw ukrytych jest więcej niż jedna. Schemat został przedstawiony na rysunku 1.



Rysunek 3. Schemat głębokich sieci neuronowych[18]

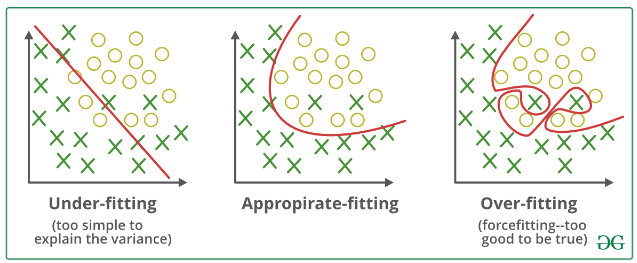
Na jej podstawie projektowany będzie model potrzebny do trenowania modelu. Warstwy, jako że mamy do czynienia z modelem sekwencyjnym będą dodawane jedna po drugiej[21]. w  tej części opisane zostaną poszczególne warstwy używane w dalszych rozdziałach pracy.

* Zadaniem warstwy *Flatten* jest „spłaszczenie” obrazu tzn. przekonwertowanie np. obrazu 32x32 na taki o wymiarach 1x1024.
* Korzystać będziemy również z warstwy *Dense*. w  naszym przypadku będzie ona miała 128 jednostek. Jest to gęsto połączona warstwa sieci neuronowej.
* *Convolutional* czyli warstwa splotowa to warstwa której zadaniem jest pobieranie danych wejściowych i tworzenia z nich map obiektów.
* *Pooling* to warstwa służąca do łączenia danych przestrzennych.
* Warstwy aktywować będziemy przy pomocy funkcji aktywacyjnej ReLu czyli rectified linear.
* Ostatnią użytą warstwę aktywować będziemy funkcją *softmax* czyli funkcję do rozkładu prawdopodobieństwa która zwróci nam tensor[22].

Warstwy te dodawanie po sobie sekwencyjnie stworzą model który wykorzystany będzie do przewidywania ognia. w  przypadku tej pracy zaproponowana została następująca kolejność warstw : *Convolutional* - *Activation Relu – Pooling - Convolutional* - *Activation Relu – Pooling - Convolutional* - *Activation Relu - Convolutional* - *Activation Relu – Pooling – Fully Connected Layer – Output,*[23] a ich implementacja została opisana w dalszej części pracy.

### Pojęcia związane z uczeniem maszynowym

W niniejszej pracy, wielokrotnie pojawiać się będą podane powyżej określenia. z pojęciem epoki (ang. Epochs) mamy do czynienia wówczas, gdy dane z używanej bazy danych zostają przekazane do sieci neuronowej tylko raz. Jednak ze względu na możliwości komputera, epoki często dzielimy na mniejsze co widać przy procesie nauczania. Aby poprawiać działanie programu oraz zwiększać skuteczność rozwiązania, używamy więcej niż jednej epoki. Dobór odpowiedniej ilości *epochs* jest bardzo istotny, ponieważ zbyt mała liczba powodować będzie niedopasowanie (ang. Underfitting) zaś zbyt duża nadmierne dopasowanie (ang. Overfitting) [24]. Działanie stworzonego programu zostało sprawdzone dla 10 i 50 *epochs*, co zostało zawarte w dalszej części.



Rysunek 4. Underfitting raz overfitting[20]

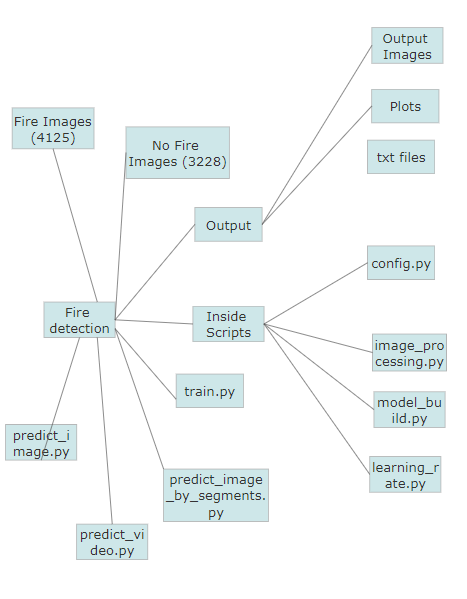
Underfitting oraz overfitting są skutkiem niskiej wydajności. Na rysunku 2. widzimy 3 przypadki dopasowania. Kółka oraz krzyżyki są reprezentacją danych użytych do szkolenia, zaś czerwona linia przedstawia predykcję w oparciu o przygotowany model. Pierwszy przykład przedstawia problem niedopasowania. Model nie jest w stanie skutecznie dostosować się do zmian i różnorodności dostarczonych danych. Problem ten występuje zwłaszcza przy zbyt małej liczbie danych, a prognozy wygenerowany przy pomocy tego modelu są często błędne. Nadmierna dopasowanie spowodowane może być nauczaniem modelu, przez zbyt duża liczbę słabej jakości danych. Wydajność spada na skutek dostarczenia zbyt wielkiej ilości szczegółów oraz innych zaburzeń. Zobrazowane zostało to na trzecim przykładzie rysunku 2., gdzie zbyt dogłębna analiza danych skutkuje problemami w przewidywaniu rezultatów próbek testowych[23].

Wielkość partii (ang. Batch Size) to zastosowany podział na jaki został podzielony zestaw danych do sieci neuronowej, aby usprawnić jego odczyt. Na potrzeby niniejszej pracy dyplomowej wartość ta została ustalona na 100. Oznacza to, że data set został podzielony, na stuelementowe podzbiory, a algorytm trenuje sieć dla każdego z nich po kolei. Zaletą stosowania takiego rozwiązania jest mniejsze zużycie pamięci operacyjnej i szybsze działanie procesów.

Na potrzeby projektu, przed procesem testowania, wyznaczony zostanie optymalna wartość parametru *learning rate*. Wyliczony zostanie on przy pomocy wbudowanej funkcji biblioteki Keras. Wskaźnik ten, został przedstawiony graficznie oraz przeanalizowany w dalszej części pracy,

# Projektowanie

Pierwszym i najważniejszym krokiem jest przygotowanie odpowiednich danych. Od tego kroku zależy jak skuteczny będzie algorytm. w  niniejszej pracy dyplomowej zaimportuję bazę obrazów liczącą ponad 6000 obrazów, a następnie podzielę ją na taką w której dochodzi do pożaru, oraz nie. Jednoznacznie określone zostanie, które zjawiska algorytm powinien wykrywać zgodnie z metodyką postępowania uczenia nadzorowanego, którego działanie opiera się na ingerencji człowieka w procesie tworzenia funkcji. Różnica pomiędzy uczeniem nadzorowanym, a uczeniem nienadzorowanym polega na tym, że w pierwszym przypadku rola człowieka polega na określeniu zależności między daną wejściową, a wyjściową, a w drugim przypadku komputer ma we własnym zakresie rozpoznać pożądany rezultat[26]. Kolejnym krokiem będzie rozdzielenie bazy na dwa zbiory danych, przeznaczonych do trenowania (ang. Training set), oraz testowania (ang. test set). w  swojej pracy pragnę nauczyć komputer samodzielnego oceniania obrazu bieżącego i sygnalizowania anomalii, a następnie przetestowania jego funkcjonalności. Gdy stworzona zostanie baza danych, należy opracować model, który wykorzystywał będzie algorytm do analizy danych[27]. Istnieje wiele różnych algorytmów. Każdy z nich ma wady i zalety oraz różne zastosowania. w  niniejszej pracy wybiorę model, dzięki któremu uzyskam najlepszą, możliwą dokładność rozpoznawania zjawiska na obrazie z kamery wizyjnej. Trenowanie odbywać będzie się w środowisku Python przy pomocy biblioteki Keras. Wówczas będę mógł przystąpić do testów, z jaką dokładnością zaimplementowany model jest w stanie rozróżnić oraz jakie parametry wpływają na skuteczność oraz specyficzność rozwiązania. w  przypadku słabych rezultatów usprawnię działanie programu przez zmianę zastosowanego modelu albo poprawę zmiennych.



Rysunek 5. Struktura projektu

Rysunek 2. przedstawia strukturę projektu. Opis poszczególnych sekcji będzie w dalszej części pracy. Pierwszym z plików jest plik konfiguracyjny *config.py.* Zawiera on ścieżki do plików oraz definicje zmiennych. z tego miejsca można łatwo dostosować projekt do wykorzystania uczenia maszynowego w celu predykcji innych danych.

Plik ten zawiera wszystkie potrzebne ścieżki oraz klasy. Zainicjalizowane są również zmienne niezbędne do prawidłowego zaimplementowania algorytmu do wykrywania ognia. Zmienne te można dowolnie modyfikować w zależności od potrzeb. Parametr *Learning Rate* został wyliczony przy pomocy osobnego skryptu przedstawionego w dalszej części pracy. Zastosowane został również podział w proporcji 4:1 danych treningowych i testowych. Przy zbyt małej liczbie danych treningowych może dojść do dużych rozbieżności w predykcji, natomiast zbyt mało danych testowych może skutkować dużą wariancją przy wynikach testu. Duża liczba obrazów zastosowanych do uczenia modelu powinna spowodować, że oczekiwany rezultat będzie wiarygodny i stabilny. Problem pojawia się przede wszystkim, w momencie gdy do trenowania zostaje użyte mało danych co zostało przedstawione w dalszej części pracy.

Ponadto na potrzeby niniejszej pracy dyplomowej stworzonych zostało kilka modeli. Różnią się one liczbą epok, liczbą obrazów użytych do trenowania oraz wejściowymi wymiarami modelu. w skrypcie *config*.py istnieje możliwość wybrania modelu, które zapisane są w folderze *inside scripts.*

## Import bazy danych

****

Rysunek 6. Przykładowe obrazu użyte do trenowania modelu

Bazę danych w niniejszej pracy dyplomowej tworzą zdjęcia i obrazy ognia w różnych środowiskach oraz zdjęcia, na których go nie ma. Zostały one podzielone na dwa katalogi. Obrazów pozytywnych wykorzystano 4125, a negatywnych 3228 Zbiór został stworzony przy użyciu zdjęć z Internetu[28],[29]. Przy doborze odpowiednich materiałów starałem się, aby były one możliwie jak najbardziej różnorodne (Rysunek 4). Zawarte zostały zarówno zdjęcia w sytuacjach kontrolowanych jak ognisko czy kominek, ale również podczas pożarów i wybuchów. Podobnie jest z obrazami nie zawierającymi objawów pożaru. Zawarte zostały zdjęcia ludzi, przyrody oraz przestrzeni miejskiej. Widok ognia zależy od wielu zewnętrznych czynników jak materiał, który ulega spalaniu i temperatura. Ponadto, w wielu przypadkach występuję również dym. Aby rezultaty były wiarygodne, wykorzystałem obrazy zawierające ogień w każdym możliwym przypadku. z racji tego, że kształt ognia jest bardzo zmienny z czasem, przy wykrywaniu zjawiska skupiłem się głównie na natężeniu kolorów. Wobec tego baza danych zawiera jedynie obrazy kolorowe.

## Przetwarzanie wstępne zdjęć oraz podział

Aby poprawić skuteczność i zwiększyć bazę danych stworzyłem skrypt odpowiedni skrypt. Wykorzystałem w tym celu bibliotekę OpenCV. Służy ona do przetwarzania obrazów. Przy jej pomocy z pobranych wcześniej zdjęć, ruchem myszką zaznaczany, a następnie wycinany jest fragment obrazu zawierający ogień. Kolejny etap generuje jego kopie obrócone o odpowiednio 90, 180 i 270 stopni. Dzięki takiej operacji końcowy zbiór danych zawiera mniej elementów tła, a zwiększona ilość obrazów generuje więcej danych wejściowych. Obrócenie ich powoduje, że przez komputer traktowane są jako zupełnie nowy zestaw informacji.

Skrypt zacząłem od dołączenia odpowiednich bibliotek niezbędnych do przetwarzania obrazów oraz zmiany folderu na zawierający wszystkie obrazy z bazy danych. Lista nazw obrazów przechowywana jest w zmiennej *filenames*, zaś *coordinates* przechowywać będzie wartości współrzędnych.

Następnie zdefiniowana została funkcja do wycinania obszaru zawierającego ogień. Interesujący fragment obrazka zaznaczany jest przy użyciu myszki. Wciśnięcie lewego przycisku definiuje pierwszy wierzchołek. Drugi odczytywany jest przez puszczenie przycisku. Aby wszystkie obrazy został przetworzone zastosowana została pętla for. w ostatniej części skryptu zawarte zostały ostateczne modyfikacje wycinka oraz zapisanie ich do pliku.

## Opracowanie modelu

Oczekiwanym ostatecznym rezultatem jest plik video z zaznaczonym miejscem, gdzie występuje zjawisko. Aby to osiągnąć należy podzielić obraz na mniejsze segmenty, a następnie dla każdego z nich przeprowadzić osobną analizę po czym scalić cały obraz.

Stworzony model przeprowadza analizę i predykcję dla segmentów w rozmiarze 32x32 dla modelu przestrzeni barw RGB oraz 2 etykiet ([Fire, Non-Fire])[30]. Obraz poddawany analizie zostaje podzielony na części, oraz sprawdzany jest warunek czy dla niego prawdopodobieństwo wystąpienia ognia jest większe od ustalonej wartości. w  przypadku powodzenia operacji zaznaczane jest to na obrazie wyjściowym. Istotnym faktem jest również zastosowanie modelu RGB, ponieważ stworzony algorytm opiera się głównie na rozpoznawaniu na podstawie natężenia barw. w  przypadku tej pracy dyplomowej liczba etykiet odpowiada stanom, które mają być sygnalizowane i wynosi 2.

W celu implementacji modelu stworzyłem osobny skrypt *training\_model.py. w*swojej pracy skorzystałem z sekwencyjnego modelu oferowanego przez Keras. Model sekwencyjny pozwala na tworzenie w sposób warstwowy i pasuje najlepiej do tego projektu, ponieważ pracuje on dla zarówno niewielu danych wejściowych i wyjściowych. Aby program działał poprawnie zaimportowałem z biblioteki Keras odpowiednie warstwy. Po zainicjalizowaniu modelu, warstwy będę dodawane po kolei.

## Trenowanie modelu

Model to matematyczna interpretacja tworzonego procesu. Aby go uzyskać należy dostarczyć zebrane dane do opracowanego wcześniej algorytmu.

### Tworzenie modelu

Skrypt *model\_build.py* odpowiedzialny jest za implementację Convolutional Neural Network (CNN), która została opisana w poprzednim rozdziale aby poprawnie przeprowadzić detekcję ognia. w  tym celu należy najpierw zaimportować odpowiedni model oraz warstwy, z których nasz algorytm będzie korzystał. Import odpowiednich struktur został w kodzie źródłowym 1.

from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import BatchNormalization  
from tensorflow.keras.layers import SeparableConv2D  
from tensorflow.keras.layers import MaxPooling2D  
from tensorflow.keras.layers import Activation  
from tensorflow.keras.layers import Flatten  
from tensorflow.keras.layers import Dropout  
from tensorflow.keras.layers import Dense

Kod źródłowy 1. Import struktur biblioteki Keras

Na potrzeby niniejszej pracy dyplomowej używać będziemy Keras Sequential API. Po przygotowaniu środowiska stworzona została klasa dla której stworzyliśmy funkcję przyjmującą jako argumenty wejściowe 4 parametry: *width, height, depth, classes.* Pierwsze trzy dotyczą wymiarów obrazu, zaś ostatni dotyczy liczby klas dla modelu. w  naszym przypadku liczba ta będzie zawsze równa 2, ponieważ istnieją dwie klasy: *Fire* oraz *Non-Fire.* Następnie inicjalizujemy model Sequential[31] oraz implementujemy zmienną *chanDim* używaną w niektórych warstwach. Cały proces dzieli się na kilka etapów. Tak jak zostało wspomniane w poprzednim podrozdziale, należy sekwencyjnie dodawać kolejne warstwy.

model.add(SeparableConv2D(16, (7, 7), padding="same", input\_shape=inputShape))  
model.add(Activation("relu"))  
model.add(BatchNormalization(axis=chanDim))  
model.add(MaxPooling2D(pool\_size=(2, 2)))

Kod źródłowy 2. Dodanie pierwszej warstwy

W kodzie źródłowym 2 dodano pierwszą warstwę *Convolutional* - *Activation Relu - Pooling* Ważnym elementem jest dodanie *input\_shape* na poziomie pierwszej warstwy. Konieczne jest, aby model znał wymiary podanego obrazu. Kolejne warstwy będą automatycznie korzystać z tej wartości.

model.add(SeparableConv2D(32, (3, 3), padding="same"))  
model.add(Activation("relu"))  
model.add(BatchNormalization(axis=chanDim))  
model.add(MaxPooling2D(pool\_size=(2, 2)))

Kod źródłowy 3. Dodanie drugiej warstwy

Dodanie kolejnej warstwy *Convolutional* - *Activation Relu – Pooling* zostało przedstawione w kodzie źródłowym 3.

model.add(SeparableConv2D(64, (3, 3), padding="same"))  
model.add(Activation("relu"))  
model.add(BatchNormalization(axis=chanDim))  
model.add(SeparableConv2D(64, (3, 3), padding="same"))  
model.add(Activation("relu"))  
model.add(BatchNormalization(axis=chanDim))  
model.add(MaxPooling2D(pool\_size=(2, 2)))

Kod źródłowy 4. Dodanie trzeciej warstwy

Dodanie kolejnych warstw *Convolutional* - *Activation Relu - Convolutional* - *Activation Relu – Pooling* ukazuję kod źródłowy 4.

model.add(Flatten())  
model.add(Dense(128))  
model.add(Activation("relu"))  
model.add(BatchNormalization())  
model.add(Dropout(0.5))  
  
model.add(Dense(128))  
model.add(Activation("relu"))  
model.add(BatchNormalization())  
model.add(Dropout(0.5))  
  
model.add(Dense(classes))  
model.add(Activation("softmax"))

Kod źródłowy 5. Dodanie kolejnych warstw

Fully Connected layers czyli warstwy ściśle połączone, a następnie aktywacja funkcją softmax, czyli funkcją do rozkładu prawdopodobieństwa, która zwraca tensor.

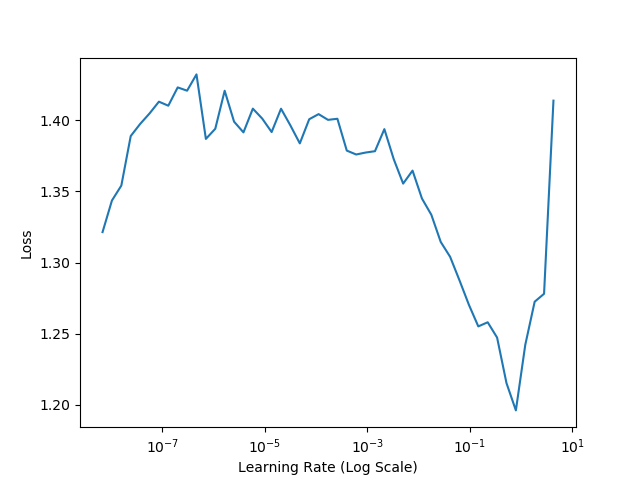
Kolejność dodawania warstw jest kluczowa do poprawnego funkcjonowania programu. Wszystkie z nich są dodawane przy pomocy funkcji *model.add.* Na końcu model jest zwracany i zostanie on użyty w dalszej części.

### Learning rate

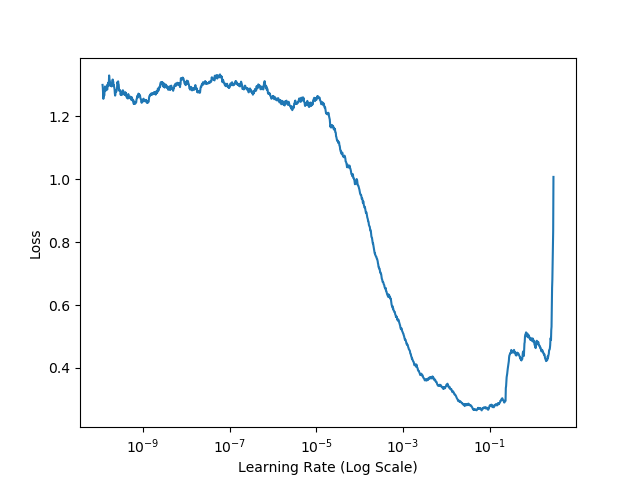
Zanim model zostanie nauczony, należy sprawdzić, dla jakiej wartości parametru *learning rate* występują najmniejsze straty. w  tym celu stworzony został skrypt. Przy jego pomocy w wybranym przez użytkownika przedziale obliczany, a następnie przedstawiany na wykresie jest pożądany parametr. Korzystać będę z wartości stanowiącej minimum na wykresie. Efekt ten otrzymany został przy pomocy funkcji oferowanej przez Keras.

Parametr ten sprawdzany został dla 4 przypadków. Rezultaty zostały przedstawione na rysunkach 3, 4, 5, 6.

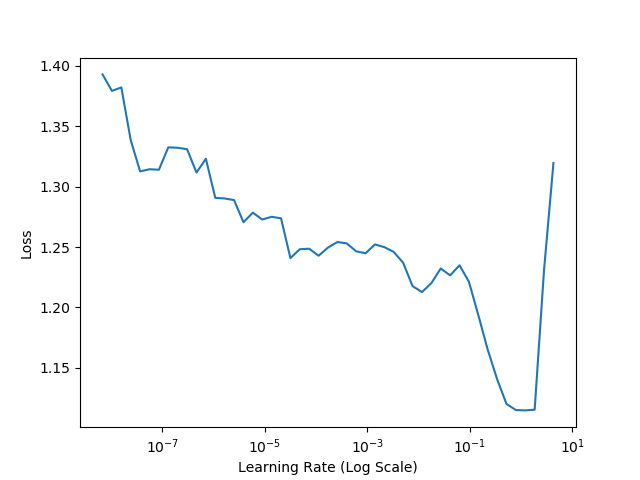
* Przypadek nr 1: liczba obrazów w  zbiorze danych :110, liczba epok : 20
* Przypadek nr 2: liczba obrazów w  zbiorze danych :4125, liczba epok : 20
* Przypadek nr 3: liczba obrazów w  zbiorze danych :110, liczba epok : 50
* Przypadek nr 4: liczba obrazów w  zbiorze danych :4125, liczba epok : 50



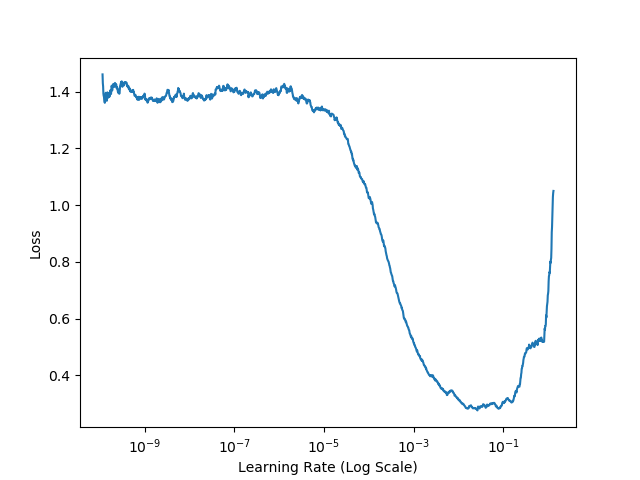
Rysunek 7. Parametr *learning rate* – przypadek I



Rysunek 8. Parametr *learning rate* – przypadek II



Rysunek 9. Parametr *learning rate* – przypadek III



Rysunek 10. Parametr *learning rate* – przypadek IV

Na podstawie powyższych wykresów można odczytać następujące wnioski. Zarówno liczba obrazów, jak i liczba epok mają wpływ na wartość parametru *learning rate*. Przypadek pierwszy do którego wykorzystano jedynie ponad 100 obrazów oraz pomiar został przeprowadzony przy liczbie epok równej 20 charakteryzuje się chaotycznościa i małą dokładnościa pomiaru. Można zaobserwować tendencję do spadku strat w przedziale wartości parametru [0.01,100], jednak nie można jednoznacznie określić jaką wartość najlepiej przyjąć, ponieważ nawet w minimum funkcji wartość strat jest duża, większa od 1, dlatego powtórzony został cały proces. Podobny wynik został otrzymany przy tej samej liczbie obrazów w zbiorze danych, lecz zwiększeniu liczbie epochs. w przypadku zbyt małej liczby obrazów, występują niepożądane ekstrema funkcji. Rezultat tego pomiaru przedstawia rysunek 9. Na rysunku 8. została ukazana próba przy liczbie obrazów wykorzystanych do szkolenia równiej 4125, jednak liczbie epok takiej samej jak w przypadku 1. oraz 3. w  przypadku dużo większej liczby dostarczonych danych, widać że wyniki są dużo bardziej stabilne. Współczynnik strat dla wartości parametru 0.01 osiąga już około 0.4 co jest zadowalającym wynikiem. Ostatnia próba została przeprowadzona dla 4125 obrazów oraz 50 epok, dla której uzyskano najlepszy i najbardziej wiarygodny rezultat. *Learning late* dla tego przypadku również oscyluje w okolicy jednej wartości – 0.01. Wartość została zapisana w skrypcie *config.py.*

### Trenowanie modelu

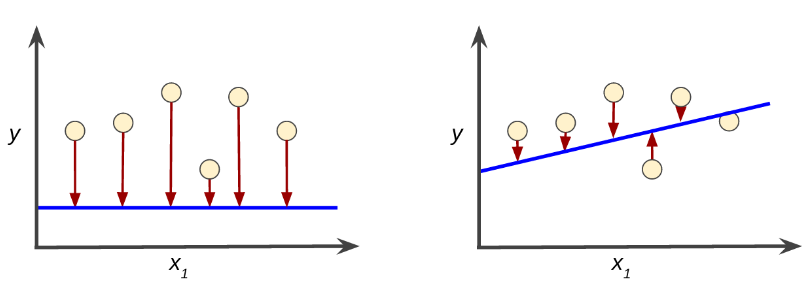
W tym celu stworzony został skrypt *train.py.* Jego zadaniem jest wykorzystanie stworzonej bazy danych do wytrenowania modelu. Zostanie zweryfikowana również poprawność podjętych dotychczas działań[32]. Początkowym krokiem jest załadowanie obrazów. Są one umieszczone w dwóch osobnych folderach, podzielone według tego, czy jest na nich ogień czy też nie. Zanim model zacznie zostać trenowany należy go skompilować (kod źródłowy 6.).

opt = SGD(lr=config.INIT\_LR, momentum=0.9, decay=config.INIT\_LR / config.NUM\_EPOCHS)  
model = model\_build.build(width=32, height=32, depth=3, classes=2)  
model.compile(loss="binary\_crossentropy", optimizer=opt, metrics=["accuracy"])

Kod źródłowy 6. Kompilacja modelu

Korzystamy z utworzonego przez nas wcześniej modelu podając mu odpowiednie parametry wejściowe. w  funkcji *compile* – oferowanej przez środowisko Keras - podajemy 3 argumenty. Pierwszy z nich *loss* czyli to, co model będzie starał się zminimalizować. Następnie korzystamy z zainicjalizowanego wcześniej optymalizatora SGD. Ostatnia z funkcji – *metrics* pomoże nam w ocenie wydajności modelu. Aby wytrenować model skorzystam z funkcji dostępnej przez bibliotekę keras *fit\_generator.* Aby móc wykorzystywać stworzony i wytrenowany model w dalszej części ćwiczenia zapiszę go, a następnie stworzę diagramy pokazujące skuteczność działania.

*Training loss* oraz *Accuraccy* zostały przedstawione na wykresach. Podobnie jak w przypadku *learning rate*, rozważane zostały 4 przypadki, w zależności od liczby *epochs*, oraz liczby danych dostarczonych modelowi.



Rysunek 11. Parametr *training loss*[25]

*Training loss* jest parametrem ukazującym straty w trenowanym modelu. Im niższy jest ten współczynnik, tym lepszy model został wytrenowany. Obliczany jest on na podstawie szkolenia oraz walidacji, a jego analiza przedstawia skuteczność działania. Jest to suma błędów, dla każdego przypadku. Rysunek 9. ukazuję po lewej stronie model generujący wysokie strany, natomiast po prawej stronie dobrze wytrenowany model, gdzie liczba strat została zminimalizowana. Przedstawione zostały na nim dane wejściowe w postaci żółtych kropek, oraz przewidywany rezultat w postaci niebieskiej kreski. Współczynnik strat został zobrazowany jako strzałki. Na podstawie powyższego rysunku, można wywnioskować, że przypadek po prawej stronie jest bardziej skuteczny, ponieważ lepiej wyszkolony model, z krzywą predykcji bardziej dostosowaną do danych wyjściowych powoduję zmniejszenie strat.

*Accuraccy* jest parametrem wyrażanym w procentach. Na podstawie wyszkolonego modelu i bazy danych testowej, sprawdzana jest precyzja rozwiązania. Training set porównuje wyniki osiągnięte przy użyciu wyszkolonego modelu, z określonymi przewidywaniami przez użytkownika. Wynikiem jest iloraz poprawnych do wszystkich prób predykcji. *Accuracy* zawiera się w przedziale 0-1 i im większa wartość, tym lepiej.



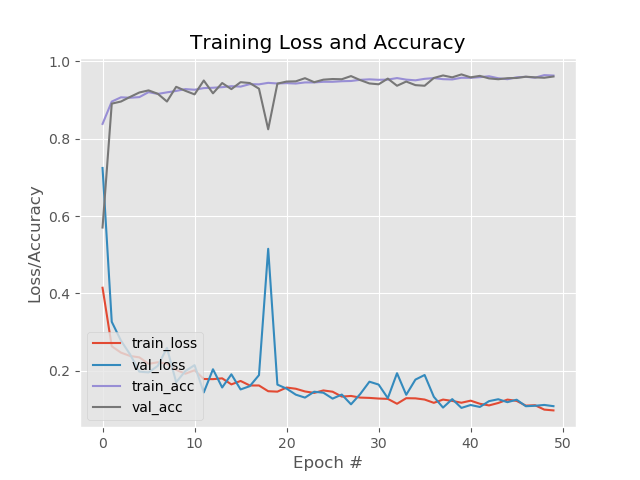
Rysunek 12. Wykres *accuracy* oraz *training loss*. Przypadek I



Rysunek 13. Wykres *accuracy* oraz *training loss*. Przypadek II



Rysunek 14. Wykres *accuracy* oraz *training loss*. Przypadek III



Rysunek 15. Wykres *accuracy* oraz *training loss*. Przypadek IV

W przypadku pierwszym, ciężko jest zaobserwować jakąś prawidłowość. Dane charakteryzują się dużą losowością i przypadkowością. w przypadku drugim zwiększona została ilość obrazów wykorzystanych do uczenia co skutkowało poprawą wyników. Wartość accuraccy dąży do 1, a parametr strat maleje wraz z każdą kolejną epoką. Przypadek trzeci, podobnie jak pierwszy nie wskazuje tendencji do monotoniczności funkcji. Wynika z tego, że na wartości badanych parametrów wpływa przede wszystkim liczba obrazów w danych szkolących, a nie liczba epochs. Najlepsze wyniki uzyskano przy ponad 4000 obrazach oraz 50 epokach co obrazuję rysunek 10., który przedstawia przypadek 4.

Na podstawie powyższych wykresów widać, że bardzo istotne jest dostarczenie trenowanemu modelowi dużej bazy danych, ponieważ przy małej ilości wyniki są bardzo nieprzekonywujące, bardzo wysoki jest współczynnik strat i ciężko dostrzec jednoznaczną tendencje. w  przypadku dużo większej liczby dostarczonych obrazów, widać że wyniki są dużo bardziej zadowalające oraz stabilne. Dokładność jest na wysokim poziomie i rośnie wraz z liczbą epok[33]. Tak więc przypadek 4 jest zadowalający i dla niego przeprowadzone zostaną testy oraz dalsze postępowania.

## Predykcja, dokładność

Skrypt *predict\_fire* pokazuje rezultaty całej pracy. Do przewidywania korzysta on z opracowanego wcześniej modelu, który należy najpierw zaimportować, aby w praktyczny sposób pokazać prawidłowe działanie algorytmu.

Proces testowania został podzielony na 3 etapy. Pierwszy z nich dotyczy analizowania całych obrazów. Model został wytrenowany w ten sposób, aby rezultatem była odpowiedź, czy prawdopodobieństwo wystąpienia ognia w podanym na wejściu obrazie jest większe niż 80%. Do testowania została użyta baza 30k obrazów[34]. Zawiera ona zdjęcia nie zawierające ognia. Przykładowe obrazy przedstawione zostały na rysunku nr 14. Jako, że są to obrazy, które nie przedstawiają zjawiska, które algorytm powinien wykrywać, oczekiwanym rezultatem jest brak informowania użytkownika o prawdopodobieństwie wystąpienia ognia powyżej 80%.



Rysunek 16. Przykładowe obrazy z bazy Flickr

Drugi etap dotyczy analizy pojedynczych obrazów jednak po podzieleniu na mniejsze segmenty. Zadaniem programu jest nie tylko wykrycie ognia, ale również wskazanie w której części się on znajduję[33].

Ostateczny efekt przedstawia film video, który jest udoskonaleniem kroku 2. Plik w formacie mp4 zostaje podzielony na mniejsze klatki, i każda klatka zostaje analizowana podobnie jak w  poprzednim punkcie. Następnie cały wraz z nałożoną maską zostaje powtórnie sklejony.

Wykorzystywany model dla trzech przypadków różni się jedynie parametrami definiującymi szerokość oraz długość obrazu wejściowego. w  przypadku całych obrazów korzystam z wymiarów 128x128, zaś przy dzieleniu na segmenty 32x32. Pierwsza część testów nastawiona jest na poprawne ocenienie czy obraz przedstawia wykrywane zjawisko. Analiza przeprowadzana jest na podstawie całego obrazu, dlatego rozdzielczość, jest większa niż dla drugiego przypadku. Wartość ta, powinna być jak największa i ograniczona jest przede wszystkim ze względu na obrazy dostarczane w training set, do których została dostosowana. Zdjęcia użyte do trenowania modelu są w różnej jakości, a opracowywanie modelu dla zbyt dużej rozdzielczości skutkowałoby przekłamaniem wyników. w przypadku drugim, gdy analiza odbywa się po wcześniejszym podziale obrazu na mniejsze elementy, wartość 32x32 jest kompromisem pomiędzy jak największą rozdzielczością dla jak najlepszych rezultatów, a jak najmniejszą dla jak najdokładniejszego wskazania epicentrum pożaru.

Pierwszy etap ma jedynie na celu pokazać odpowiednie dobranie baz danych, zaś przy podziale na segmenty pragnę uzyskać dokładniejszy wynik, kosztem szybkości działania. Obraz w pozostałych dwóch przypadkach przeskalowany będzie do wielkości 256x256 a następnie analizowany przez mniejsze elementy 32x32 w konsekwencji czego każdy z nich podzielony będzie na 64 mniejsze segmenty.

### Analiza całych obrazów

Do testowania pierwszej serii danych wykorzystam model przystosowany do parametrów wejściowych 256x256. Obrazy przeskalowane zostaną do tych samych rozmiarów. Danymi wyjściowymi będzie wynik przewidywania z przedziału [0,1]. Gdzie 1 oznacza 100% pewność wystąpienia ognia. Program do predykcji został skonfigurowany w ten sposób, aby informował użytkownika gdy wynik przekroczy 0.9, ponieważ oznacza to, że prawdopodobieństwo wystąpienia ognia wynosi ponad 90%. Fragment kodu został przedstawiony w kodzie źródłowym 7. Wszystkie obrazy użyte do testowania nie zawierają ognia, zatem przewidywanym rezultatem jest stan, w którym program nie poinformuje o wykryciu zjawiska.

if predictions[1] > 0.9:  
 print(imagePath)

Kod źródłowy 7. Sprawdzenie prawdopodobieństwa

Przedstawiony powyżej fragmentu kodu, który jest częścią skryptu *predict\_image.py,* sprawdza przy użyciu modelu jakie jest prawdopodobieństwo ognia. Zmienna *predictions* przechowuje dwie wartości. *Predictions[0]* jest prawdopodobieństwem klasy *No-Fire*, zaś *Predictions[1]* prawdopodobieństwem klasy *Fire.* Suma obu wartości wynosi 1. Informacja o wystąpieniu ognia zostaje podana, przy *Predictions[1]* >0.9 czyli przy prawdopodobieństwie większym niż 90%. Przeanalizowanych zostało 31 783 obrazów. Dla 310 z nich program wskazał wykrycie ognia. Wynika z tego, że pomyłka nastąpiła jedynie dla 0.975%. Zaś 99.025% obrazów zostało poprawnie ocenionych. Rezultat tego testu jest pozytywny, ponieważ skuteczność poprawnego rozpoznawania ognia jest na wysokim poziomie, bliskim 100%. Kolejnym etapem będzie testowanie pojedynczych obrazów segment po segmencie.

### Analiza pojedynczych fragmentów obrazów

Proces predykcji przebiega w ten sam sposób. Korzystając z nauczonego wcześniej modelu program ma za zadanie określić z jakim prawdopodobieństwem wykryte zostanie zagrożenie.

Jedyną różnicą jest to, że obraz nie jest analizowany w całości, lecz fragment po fragmencie. Wysoki wynik, ponad 80% szansy na wystąpienie ognia został oznaczony przez naniesienie czerwonego prostokąta, co widoczne jest na rysunku poniżej. W tej części pracy dyplomowej korzystał będę ze skryptu *predict\_image\_by\_segments.py.* Aby pokazać skuteczność działania przedstawione zostały dwa przykładowe zdjęcia. Obraz wejściowy jest wyżej, zaś otrzymany efekt niżej. Zgodnie z oczekiwaniami, oba zdjęcia różnią się od siebie o naniesioną maskę

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 98 | 97 | 97 | 96 | 96 | 96 | 95 | 96 | 96 | 97 | 97 | 98 | 96 | 97 | 97 | 98 |
| 96 | 95 | 94 | 92 | 89 | 87 | 85 | 88 | 87 | 90 | 91 | 93 | 91 | 93 | 94 | 95 |
| 92 | 89 | 89 | 76 | 74 | 82 | 87 | 88 | 75 | 77 | 73 | 89 | 87 | 86 | 88 | 91 |
| 88 | 85 | 87 | 1 | 0 | 9 | 69 | 85 | 82 | 77 | 81 | 88 | 85 | 84 | 84 | 86 |
| 85 | 84 | 87 | 1 | 0 | 0 | 0 | 6 | 86 | 83 | 76 | 79 | 85 | 85 | 84 | 84 |
| 86 | 87 | 88 | 0 | 0 | 0 | 1 | 20 | 69 | 82 | 71 | 82 | 89 | 87 | 87 | 87 |
| 88 | 89 | 90 | 0 | 0 | 0 | 10 | 9 | 5 | 78 | 74 | 84 | 90 | 89 | 90 | 90 |
| 90 | 92 | 96 | 0 | 0 | 1 | 3 | 12 | 0 | 18 | 75 | 80 | 96 | 96 | 95 | 93 |
| 96 | 97 | 98 | 0 | 0 | 3 | 3 | 9 | 10 | 79 | 82 | 84 | 98 | 97 | 97 | 96 |
| 98 | 99 | 99 | 96 | 0 | 16 | 57 | 20 | 86 | 84 | 86 | 87 | 99 | 99 | 98 | 98 |
| 99 | 99 | 99 | 90 | 12 | 1 | 86 | 63 | 92 | 89 | 92 | 91 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 98 | 97 | 3 | 89 | 90 | 87 | 85 | 91 | 96 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 97 | 97 | 78 | 79 | 90 | 95 | 96 | 91 | 96 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 98 | 98 | 98 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 98 | 99 | 99 | 97 | 96 | 92 | 94 | 96 | 97 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 98 | 96 | 95 | 94 | 89 | 91 | 95 | 97 | 99 | 99 | 99 | 99 | 99 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Rysunek 17. Macierz prawdopodobieństwa



Rysunek 18. Przykładowy obraz zawierający ogień

Aby jeszcze lepiej przekazać istotę rozwiązania i udowodnić poprawność przeprowadzonych operacji poniżej przedstawiona została macierz prawdopodobieństw dla rysunku zawierającego ogień:

Wynik tego testu bardzo dobrze obrazuję zasadę działania całego algorytmu oraz pokazuje jego skuteczność. Cały obraz został podzielony na mniejsze segmenty, dla których osobno zostały wyliczone prawdopodobieństwa. Miejsca w których znajduje się ścisłe centrum występowania ognia jest jednoznacznie określone i na potrzeby zrozumienia zostało oznaczone kolorem czerwonym. Podobnie miejsca w których nie występuje zagrożenie, które są w kolorze czarnym. Miejsca przejściowe oscylują pomiędzy skrajnymi wartościami 0 i 1, jednak jest to uzasadnione ponieważ w  tych częściach obrazu może występować połowiczne lub częściowe pojawienie się ognia.

Kolejnym przeprowadzonym przeze mnie testem, jest sprawdzenie funkcjonalności na pliku video w formacie mp4. Proces predykcji jest bardzo podobny do opisanego w poprzednim podrozdziale. Film zostaje podzielony na klatki, a każda klatka zostaje poddana weryfikacji, analogicznie do obrazów analizowanych uprzednio. Aby taki efekt otrzymać na video, plik w formacie mp4 został odczytany przy pomocy funkcji *VideoCapture, a*następnie odczytany funkcją *read* (obie funkcje dostępne są w bibliotece *cv2),* co zostało przedstawione poniżej w kodzie źródłowym 8.

vidcap = cv2.VideoCapture('video2.mp4')  
success ,image = vidcap.read()

Kod źródłowy 8. Odczyt pliku video

Wynikiem tych działań otrzymaliśmy klatkę z filmu, dla której, podobnie jak dla obrazu zostanie przeprowadzona analiza, a następnie funkcją *cv2.VideoWriter* przeprowadzone zostało ponowne złączenie do formatu video. Analiza video wykorzystuje skrypt *predict\_video.py.*

Tabela 1. Numeryczne podsumowanie wyników

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | tRAINING LOSS | VALIDATION LOSS | TRAINING ACCURACY | VALIDATION ACCURACY |
| Przypadek 1. | 0.388 | 1.564 | 0.839 | 0.520 |
| Przypadek 2. | 0.182 | 0.208 | 0.938 | 0.945 |
| Przypadek 3. | 0.240 | 1.603 | 0.889 | 0.557 |
| Przypadek 4 | 0.075 | 0.085 | 0.989 | 0.988 |

### Test czułości rozwiązania

Ostateczna weryfikacja ma na celu sprawdzenie czułości rozwiązania. Wykorzystany został model z rozdziału 5.5.1 do analizy całych obrazów. Testy zostały wykonane dla nowej, nieużytej do wcześniejszego trenowania bazie danych. Dane zagadnienie najlepiej obrazuje macierz pomyłek, ukazana w tabeli 2. W celu jej utworzenia wykorzystano 1000 obrazów zawierających ogień, oraz taka sama liczba obrazów, na których nie zarejestrowano pożaru[36].

Tabela 2. Macierz pomyłek

|  |  |  |
| --- | --- | --- |
|  | Ogień (Przewidziany) | Brak Ognia (Przewidziany) |
| Ogień (Zadeklarowany) | **True Positive** | **False Negative** |
| Brak ognia (Zadeklarowany) | **False Positive** | **True Negative** |

Powyższa tabela zawiera określenia odnoszące się do rezultatów prognoz przeprowadzonych przy użyciu wytrenowanego modelu.

* True Positive – Prawidłowe wykrycie ognia
* False Postive – Nieprawidłowe wykrycie ognia
* False Negative – Nieprawidłowe wykrycie braku ognia
* True Negative – Prawidłowe wykrycie braku ognia

Na podstawie powyższych wartości można obliczyć wartości parametrów *Accuracy, Recall, Precision, F-Measure.* Formuły do obliczania zostały przedstawione poniżej.

* Accuracy: jest to stosunek sumy True Positive i True Negative do sumy wszystkich przypadków
* *Recall:* jest to stosunek True Postive do sumy True Positive i False Negative
* *Precision:* jest to stosunek True Postitive do sumy True Positive i False Positive
* *F-Measure:* jest to stosunek iloczynu parametru *Recall*  i *Precision* do sumy *Recall* i *Precision* pomnożony dwukrotnie.

Wartości uzyskane przy użyciu wytrenowanego modelu zostały przedstawione w tabeli 3., a następnie obliczono wartości parametrów.

Tabela 3. Macierz pomyłek dla wytrenowanego modelu

|  |  |  |
| --- | --- | --- |
|  | Ogień (Przewidziany) | Brak Ognia (Przewidziany) |
| Ogień (Zadeklarowany) | **970** | **30** |
| Brak ognia (Zadeklarowany) | **78** | **922** |

* *Accuracy* : (970+922)/(970+922+30+78) = 0.946
* *Recall* : 970/(970+30) = 0.970
* *Precision* : 970/(970+78) = 0.925
* *F-Measure* : 2\*0.970\*0.925/(0.970+0.925) = 0.946

Powyższa tabela prezentuje ciekawe wnioski. Wartość False Positive jest dużo większa niż wartość False Negative, czyli istnieje większa szansa na zasygnalizowanie błędnego wystąpienia ognia, niż brak informacji w przypadku pożaru. Wynika to z faktu, że jako ogień, przez program klasyfikowane mogą być również zjawiska jak zachód słońca, czy jesienna flora.

Może się nasunąć również pytanie, dlaczego skuteczność tego rozwiązania jest gorsza niż w przypadku 5.5.1, skoro wykorzystywano ten sam model. Poprzedni test, informował o ogniu przy bardzo wysokim prawdopodobieństwie, powyżej 90%, ponieważ planowano osiągnąć algorytm, który skupi się na skuteczności rozwiązania. Skupiając się na czułości, zastosowano prób prawdopodobieństwa 50%, ponieważ wyniki potrzebne do analizy musiały zostać przedstawione zero-jedynkowo.

# Podsumowanie

Uczenie maszynowe znajduję zastosowanie w wielu dziedzinach życia codziennego. Algorytm stworzony na potrzeby niniejszej pracy dyplomowej, może zostać używany, aby usprawnić działanie systemów przeciwpożarowych.

Teoretyczne wprowadzenie do uczenia maszynowego pozwoliło wykorzystać wiedzę w praktyce, do zaimplementowania odpowiedniego algorytmu. Cały program został stworzony w środowisku, które najlepiej spełnia oczekiwania w dziedzinie Deep Learning. Proces nauczania modelu przebiegł pomyślnie, co wnioskować można po załączonych wykresach. Czułość oraz dokładność są na bardzo wysokim poziomie, a straty maleją wraz z każda *epochs*.

Jednak najlepszym wyznacznikiem skuteczności działania, są zdjęcia na których dokładnie widać poprawność wykonywanych operacji. Zdjęcia zawierające ogień jednoznacznie ukazują miejsce, gdzie on się znajduje, zaś pozostałe obrazy z dużą dokładnością nie wykazują zdolności do pomyłek.

Zaproponowane rozwiązanie spełnia swoje założenia. Poprawność wykonywanych predykcji jest bardzo wysoka. w przypadku analizy ponad 30 tysięcy obrazów, jedynie niecały jeden procent wykrył anomalie. Pod uwagę brana była nie tylko czułość, ale również specyficzność rozwiązania. Sprawdzona została w ten sposób zdolność do nie wykrywania fałszywych alarmów (ang. False Positive).

# Bibliografia

[1]Python Documentation, https://docs.python.org/, (dostęp: 22.10.2019r.)

[2]Keras Documentation, https://keras.io/, (dostęp: 20.10.2019r.)

[3]Tensorflow API Documentatio, https://www.tensorflow.org/api\_docs, (dostęp: 31.10.2019r.)

[4]Pandas: powerful Python data analysis toolkit, https://pandas.pydata.org/pandas-docs/stable/, (dostęp 22.10.2019r.)

[5]scikit-learn Machine Learning in Python, https://scikit-learn.org/stable/, (dostęp: 22.10.2019r.)

[6]Which programming language is the best for machine learning and artificial intelligence, https://www.blog.duomly.com/which-programming-language-is-the-best-for-machine-learning-and-artificial-intelligence/, (dostęp: 01.02.2020r.)

[7]Programming languages, libraries and frameworks, https://blog.alexdevero.com/programming-languages-libraries-and-frameworks/, (dostęp: 01.02.2020r.)

[8]Introduction to Keras, https://link.springer.com/chapter/10.1007/978-1-4842-2766-4\_7, (dostęp: 18.11.2019r.)

[9]Microsoft Cognitive Toolkit(CNTK), an open source deep-learning toolkit, https://github.com/microsoft/CNTK, (dostęp: 16.12.2019r.)

[10] Theano Documentation, http://deeplearning.net/software/theano/, (dostęp: 16.12.2019r.)

[11] Why use keras?, https://keras.io/why-use-keras, (dostęp: 01.02.2020r.)

[12] Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow : pojęcia, techniki i narzędzia służące do tworzenia inteligentnych systemów, Helion, Gliwice 2018

[13] TensorFlow or Keras? Which one should I learn?, https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0, (dostęp: 12.11.2019r.)

[14] Open Source Computer Vision- documentation, https://opencv.org/, (dostęp: 22.10.2019r.)

[15] Numpy Documentation, https://numpy.org/doc/, (dostęp: 22.10.2019r.)

[16] Jak wybrać algorytmy dla Azure Machine Learning Studio, https://docs.microsoft.com/pl-pl/azure/machine-learning/studio/algorithm-choice, (dostęp: 29.10.2019r.)

[17] John Hearty, Zawwansowane uczenie maszynowe z językiem Python, Helion, Gliwice 2017

[18] Convolutional neural network, https://en.wikipedia.org/wiki/Convolutional\_ neural\_network, (dostęp: 12.12.2019r.)

[19] Antonio Gulli, Sujit Pal, Deep learning with Keras, Packt Publishing, Birmingham 2017

[20]Underfitting and Overfitting in Machine Learning, https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/, (dostęp: 01.02.2020r.)

[21] Peter Flach, Machine learning : the art and science of algorithms that make sense of data, Cambridge University Press, Cambridge 2012

[22]Fire and smoke detection with Keras and Deep Learning, https://www.pyimagesearch.com/2019/11/18/fire-and-smoke-detection-with-keras-and-deep-learning/, (dostęp: 5.11.2019r.)

[23]Overfitting and underfitting with machine learning algorithms, https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/ , (dostęp: 01.02.2020r.)

[24] Epochs vs iterations vs batch size, https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 (dostęp: 16.12.2019r.)

[25]Descending into ML: Training and Loss, https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss, (dostęp: 01.02.2020r.)

[26] Chris Albon, Uczenie Maszynowe w Pythonie :receptury, wyd. Helion, Gliwice 2019

[27] Your first machine learning project in python step-by-step, https://machinelearningmastery.com/machine-learning-in-python-step-by-step/?fbclid=IwAR0sDN7vd7QDFq4omdXnP9pV3EYAmDJOdxpjgDp07ocSw0eJQdJKyJruHAA, (dostęp: 16.10.2019r.)

[28] Azure AI Gallery,https://gallery.azure.ai/, (dostęp: 10.11.2019r.)

[29] Fire Detection Image Dataset https://github.com/cair/Fire-Detection-Image-Dataset?fbclid=IwAR2b-IIqbTS2bdZL1URbh3gpeRbTJNuIVdKjNE\_53ICZOa3UpqeqCNZK\_Ds, (dostęp: 10.10.2019r.)

[30] Python machine learning mini course, https://machinelearningmastery.com/python-machine-learning-mini-course/, (dostęp: 20.11.2019r.)

[31] 3 ways to create a Keras model with TensorFlow 2.0 (Sequential, Functional and Model Subclassing), https://www.pyimagesearch.com/2019/10/28/3-ways-to-create-a-keras-model-with-tensorflow-2-0-sequential-functional-and-model-subclassing/, (dostęp: 5.11.2019r.)

[32] Keras Tutorial: The ultimate beginner’s guide to deep learning in python, https://elitedatascience.com/keras-tutorial-deep-learning-in-python, (dostęp: 30.11.2019r.)

[33] Keras cctv fire detection, https://github.com/YunHo0325/Meerkat/blob/master/README.md, (dostęp: 22.11.2019r.)

[34] flickr image dataset, https://www.kaggle.com/hsankesara/flickr-image-dataset, (dostęp: 01.12.2019r.)

[35] Experimentally Defined Convolutional Neural Network Architecture Variants for Non-temportal Real-time Fire Detection, https://github.com/tobybreckon/fire-detection-cnn?fbclid=IwAR2AvXTiZWc28xA65dvQ81TZQXGMDiClwr9x-0Uw3Xb-C1Hec7\_JgfwnzMQ, (dostęp: 20.11.2019r.)

[36]Confusion matrix machine learning, https://www.geeksforgeeks.org/confusion-matrix-machine-learning/, (dostęp: 15.02.2020r.)

Spis ilustracji:

Rysunek 1. Tradycyjne podejście[12]

Rysunek 2. Podejście wykorzystujące uczenie maszynowe[12]

Rysunek 3. Schemat głębokich sieci neuronowych

Rysunek 4. Underfitting raz overfitting

Rysunek 5. Struktura projektu

Rysunek 6. Przykładowe obrazu użyte do trenowania modelu

Rysunek 7. Parametr *learning rate* – przypadek I

Rysunek 8. Parametr *learning rate* – przypadek II

Rysunek 9. Parametr *learning rate* – przypadek III

Rysunek 10. Parametr *learning rate* – przypadek IV

Rysunek 11. Parametr *training loss*

Rysunek 12. Wykres *accuracy* oraz *training loss*. Przypadek I

Rysunek 13. Wykres *accuracy* oraz *training loss*. Przypadek II

Rysunek 14. Wykres *accuracy* oraz *training loss*. Przypadek III

Rysunek 15. Wykres *accuracy* oraz *training loss*. Przypadek IV

Rysunek 16. Przykładowe obrazy z bazy Flickrtra

Rysunek 17. Macierz prawdopodobieństwa

Rysunek 18. Przykładowy obraz zawierający ogień

Spis kodu źródłowego:

Kod źródłowy 1. Import struktur biblioteki Keras

Kod źródłowy 2. Dodanie pierwszej warstwy

Kod źródłowy 3. Dodanie drugiej warstwy

Kod źródłowy 4. Dodanie trzeciej warstwy

Kod źródłowy 5. Dodanie kolejnych warstw

Kod źródłowy 6. Kompilacja modelu

Kod źródłowy 7. Sprawdzenie prawdopodobieństwa

Kod źródłowy 8. Odczyt pliku video

Spis tabel:

Tabela 1. Numeryczne podsymowanie wyników

Tabela 2. Macierz pomyłek

Tabela 3. Macierz pomyłek dla wytrenowanego modelu