"Uczenie maszynowe w analizach big data dużych serii pomiarowych danych metrologicznych"

(Samouczek)

Spis treści

Wprowadzenie

Rozpoczęcie nauki i wykorzystania narzędzi AI do klasyfikacji w środowisku MATLAB jest uzasadnione zarówno z perspektywy dydaktycznej, jak i inżynierskiej. MATLAB oferuje spójne i wysokopoziomowe środowisko, w którym cały pipeline — od wczytania i wstępnego przetwarzania danych, przez ekstrakcję cech, aż po trening, walidację i testowanie modeli — może być realizowany w sposób zintegrowany i powtarzalny. Dzięki takim narzędziom jak Classification Learner czy Deep Network Designer możliwe jest szybkie prototypowanie modeli bez konieczności implementacji algorytmów od podstaw. Dodatkowo graficzne interfejsy użytkownika dostępne we wspomnianych narzędziach pozwalają trenować modele AI bez konieczności pisania kodu. W efekcie znacząco obniża się próg wejścia do zagadnień AI, co pozwala projektantowi skupić się na interpretacji wyników oraz jakości danych.

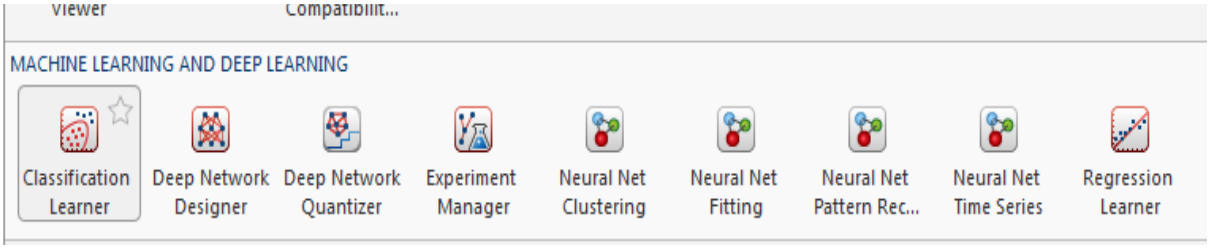
Istotną zaletą jest również możliwość łatwego przejścia od analizy danych do implementacji w systemach rzeczywistych, co ma szczególne znaczenie w zastosowaniach inżynierskich. MATLAB umożliwia integrację modeli AI z rzeczywistymi sygnałami pomiarowymi, systemami sterowania oraz środowiskiem Simulink, co pozwala na tworzenie zamkniętych pętli decyzyjnych. Dodatkowo dostępne są mechanizmy automatycznej generacji kodu, dzięki którym wytrenowane modele mogą być wdrażane na urządzeniach docelowych, takich jak mikrokontrolery czy systemy embedded.

Kolejnym istotnym aspektem jest szeroki zestaw wbudowanych algorytmów klasyfikacji — od metod klasycznych (SVM, drzewa decyzyjne, ensemble) po sieci neuronowe — oraz narzędzi do ich oceny, takich jak macierze pomyłek, krzywe ROC czy analiza błędów. Pozwala to na świadome porównywanie metod, ocenę ich zdolności generalizacji oraz dobór optymalnego rozwiązania dla konkretnego problemu. MATLAB wspiera również pracę na danych rzeczywistych, często nieidealnych (braki danych, szumy), udostępniając szereg narzędzi do obróbki i analizy danych przed procesem uczenia.

W rezultacie MATLAB stanowi efektywne i dojrzałe środowisko do nauki oraz wdrażania metod klasyfikacji opartych na sztucznej inteligencji, szczególnie w kontekście zastosowań technicznych, energetycznych i przemysłowych, gdzie istotna jest zarówno dokładność modeli, jak i ich integracja z istniejącą infrastrukturą.

Narzędzia AI w Matlab

Podstawowe narzędzia AI w MATLAB są zorganizowane w formie wyspecjalizowanych toolboxów oraz aplikacji wspomagających szybkie prototypowanie. Narzędzia te dostępne są w zakładce APPS/Machine Learning and Deep Learning (widok zakładki pokazano na rysunku 1).



Rys. 1 Widok zakładki **APPS/ Machine Learning and Deep Learning**

Najważniejsze z narzędzi AI w środowisku Matlab to:

- **Classification Learner** – aplikacja do klasyfikacji danych przy użyciu metod uczenia maszynowego (SVM, drzewa, k-NN, ensemble). Umożliwia szybkie trenowanie, porównywanie modeli oraz ocenę jakości bez konieczności programowania.
- **Deep Network Designer** – środowisko graficzne do projektowania i modyfikacji sieci neuronowych. Umożliwia m.in. wykorzystanie sieci pretrained (transfer learning) oraz wizualne budowanie architektury.
- **Regression Learner** – analogiczne narzędzie do Classification Learner, ale przeznaczone do problemów regresyjnych, pozwalające modelować zależności między zmiennymi i przewidywać wartości ciągłe.

W dalszej części opracowani przedstawione zostaną narzędzia Classification Learner i Deep Network Designer. Pokazane zostaną podstawowe kroki postępowania w przypadku każdego z narzędzi oraz proste przykłady pokazujące sposób przygotowania danych wejściowych i trenowania modelu.

Classification Learner

Classification Learner to interaktywna aplikacja w MATLAB-ie (część *Statistics and Machine Learning Toolbox*), służąca do budowy, trenowania i porównywania modeli klasyfikacyjnych bez konieczności pisania kodu. Jest to narzędzie szczególnie użyteczne na etapie eksploracji danych i szybkiego prototypowania modeli. Aplikację można uruchomić klikając w ikonę dostępną w panelu **APPS/ Machine Learning and Deep Learning**, (rys. 1) lub wpisując w command window komendę **classificationLearner**.

Narzędzie oferuje szeroki zestaw wbudowanych algorytmów klasyfikacji, m.in.:

- drzewa decyzyjne,
- maszyny wektorów nośnych (SVM),
- k-najbliższych sąsiadów (k-NN),
- modele ensemble (np. Random Forest, Boosting),
- klasyfikatory liniowe i logistyczne,
- proste sieci neuronowe.

Modele mogą być trenowane pojedynczo lub w trybie automatycznym (tzw. „Train All"), co pozwala szybko porównać wiele metod na tym samym zbiorze danych. Istotnym elementem jest możliwość walidacji krzyżowej (cross-validation), dzięki której uzyskuje się bardziej wiarygodną ocenę jakości modelu i ogranicza się ryzyko przeuczenia (overfittingu).

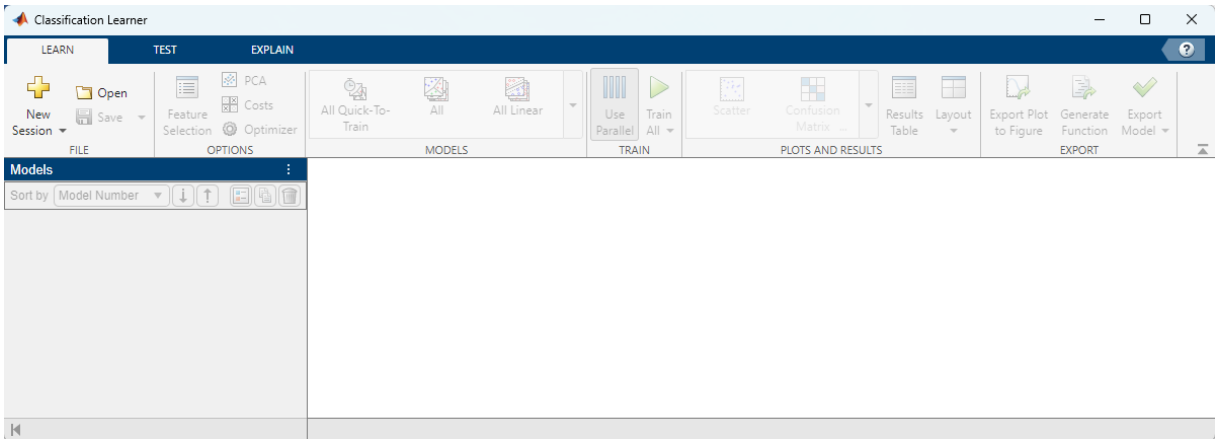
Wyniki prezentowane są w czytelnej formie:

- dokładność (accuracy),
- macierz pomyłek (confusion matrix),
- wykresy ROC,

- analiza błędnych klasyfikacji.

Użytkownik może także dostrajać hiperparametry modeli (np. głębokość drzewa, parametr C w SVM, liczba sąsiadów w k-NN) oraz obserwować wpływ tych zmian na wyniki.

Widok okna aplikacji ClassificationLearner pokazano na rysunku 2. Pracę rozpoczyna się tworząc nowa sesję (klikając w przycisk „ New Sesion) lub otwierając wcześniej zapisaną (Open).

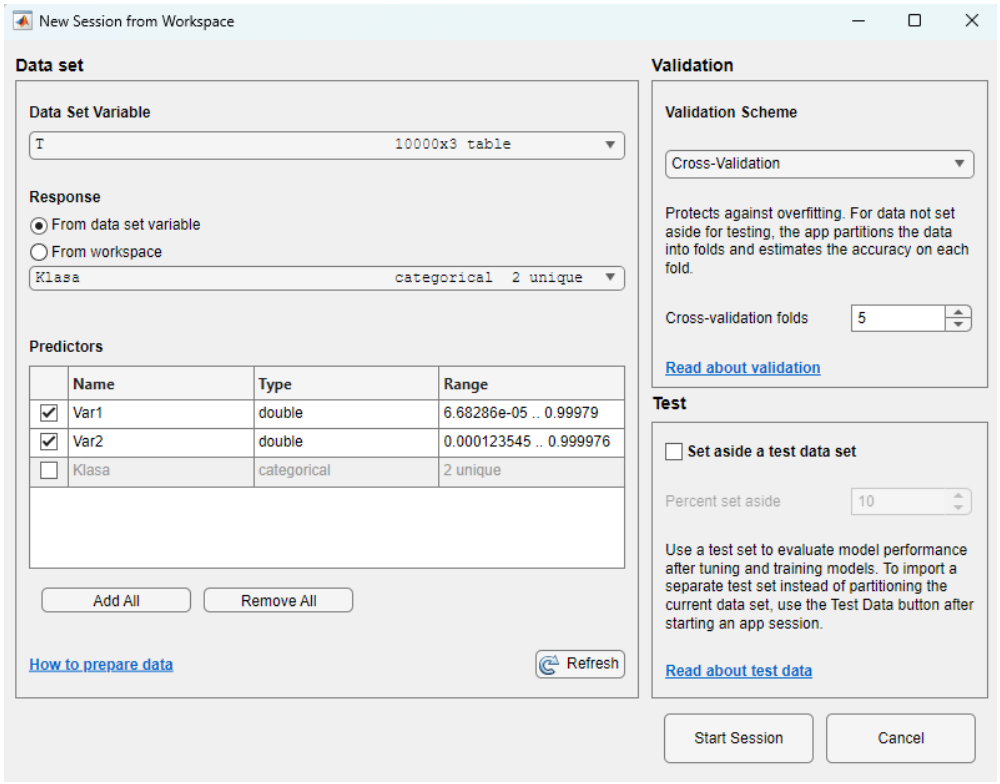


Rys. 2 Widok startowy okna aplikacji ClassificationLearner

Podstawą pracy w aplikacji jest wczytanie danych w formie tabeli (table), gdzie kolumny reprezentują cechy (Predictors), a jedna z kolumn stanowi etykietę klasy (Response). Użytkownik może wybrać, które zmienne są wejściowe, a która zmienna jest zmienną decyzyjną.

Widok okna wyboru danych pokazano na rysunku 3. W przedstawionym przykładzie dane służące do klasyfikacji umieszczono w zmiennej „T”. Klasyfikator przyjmujący dwie wartości „dobre” lub „złe” umieszone został w kolumnie o nazwie Klasa (T.Klasa).

Na etapie wczytywania danych, możliwe jest również wstępne przetwarzanie danych, takie jak normalizacja, standaryzacja czy usuwanie zmiennych.



Rys. 3 Widok okna rozpoczęcia sesji i wyboru danych to treningu

Poza wybraniem danych treningowych, użytkownik musi wybrać w jaki sposób dane zostaną podzielone na zbiory uczące i testowe w celu oceny jakości modelu (parametr **Validation Scheme).**

Dostępne są typowo trzy podstawowe schematy:

1. **Holdout Validation**

   Dane są jednorazowo dzielone na:

   - zbiór treningowy (np. 70–80%),
   - zbiór testowy (pozostała część).

   Model uczy się na jednej części danych i jest oceniany na drugiej. Jest to szybkie, ale wynik może zależeć od konkretnego podziału danych.

2. **Cross-Validation**

   Dane dzielone są na *k* części (foldów), np. 5 lub 10.
   Proces wygląda tak:

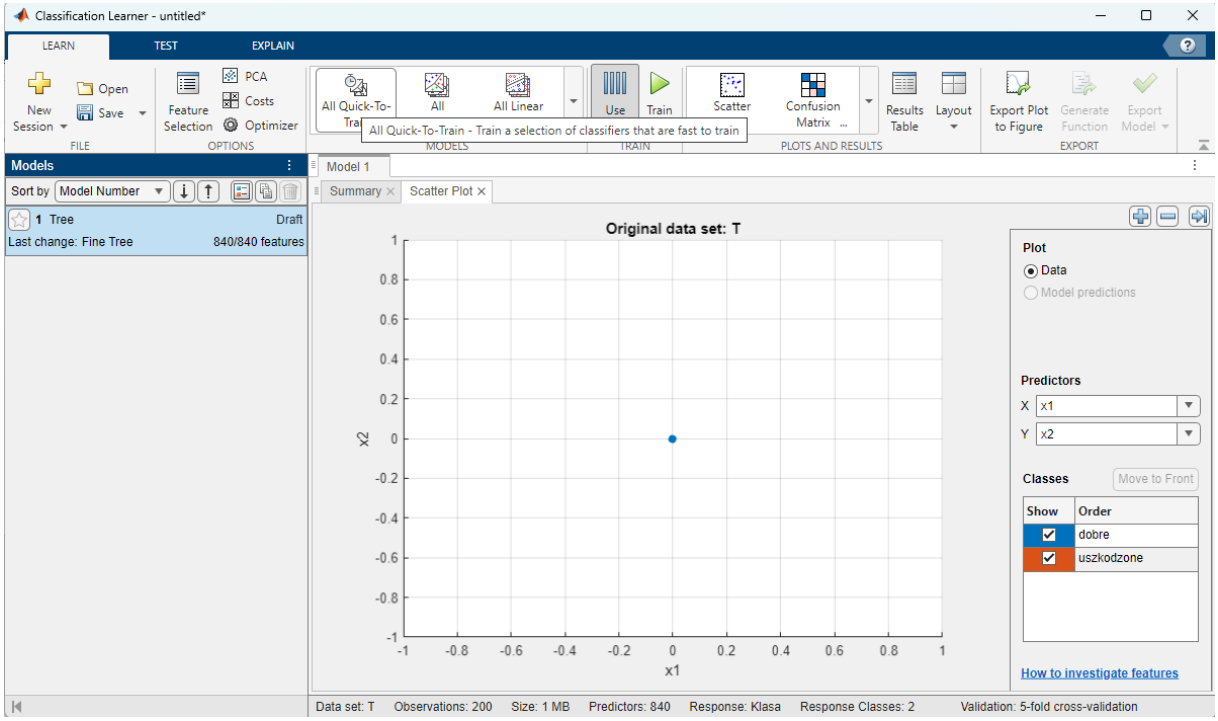
   - model trenowany jest *k* razy,
   - za każdym razem inna część jest używana jako testowa,
   - wynik końcowy to średnia z wszystkich prób.
     To podejście daje **bardziej wiarygodną ocenę**, szczególnie dla małych zbiorów danych.

3. **Resubstitution Validation**

Model jest trenowany i testowany na tych samych danych. Daje to zazwyczaj bardzo wysoką dokładność, ale jest to wynik **optymistyczny i nierealistyczny**, bo nie sprawdza generalizacji. Stosowany głównie do szybkiej diagnostyki.

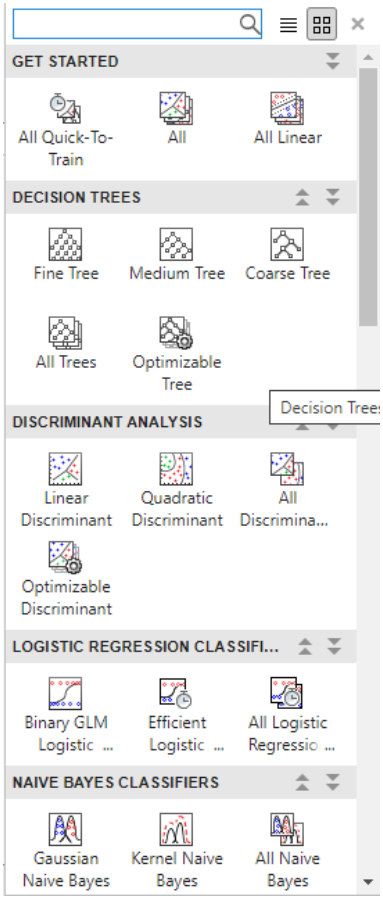
Po wczytaniu danych i rozpoczęciu nowej sesji, otwiera się okno aplikacji przedstawione na rysunku 4.

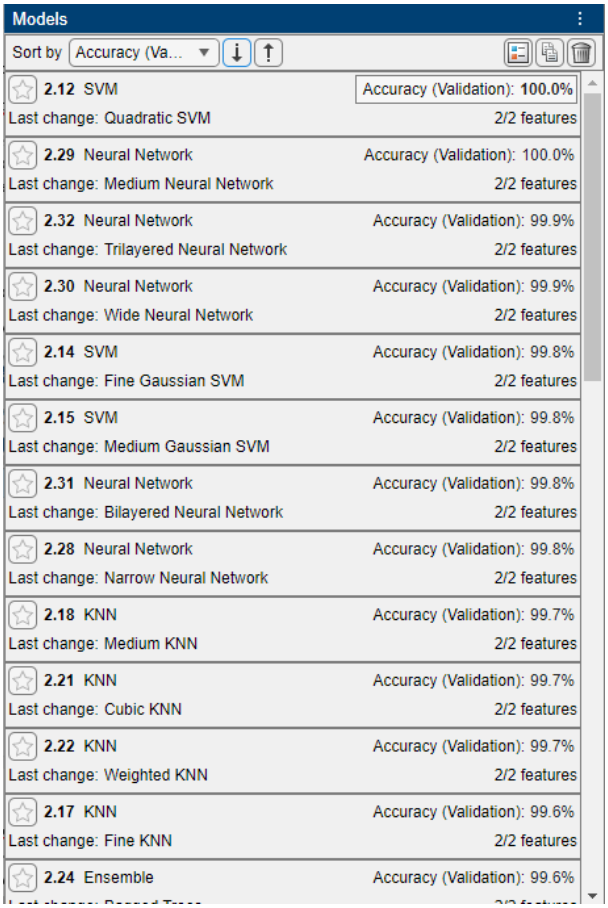
W oknie uaktywnione są dostępne możliwości operowania na danych uczących. W tym momencie użytkownik musi wybrać model, który chce zastosować i wytrenować. Możliwe jest wybranie kilku modeli klikając w ikonę modeli w oknie MODELS (rys. 5 a). Możliwe jest również wybranie wszystkich modeli szybkich do uczenia (All Quick-To-Train) lub po prostu wszystkich modeli (All).



Rys. 4 Okno aplikacji Clasyfication Learner

Trenowanie większej liczby modeli pozwala na koniec porównać wyniki między modelami i daje użytkownikowi możliwość wyboru najlepszego rozwiązania. Oczywiście wybranie większej liczby modeli wymaga więcej czasu i zasobów komputera na trening. Po wyborze dostępnych modeli należy rozpocząć proces treningu klikając ikonę Train.





Rys. 5 a) Okno wyboru modeli do trenowania, b) Okno modeli po zakończonym treningu

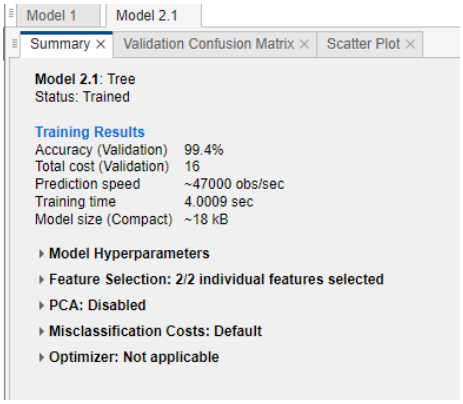
W oknie MODELS (Rys. 5 b) dostajemy informację o wynikach treningu i weryfikacji uzyskanych modeli. Poza tym można wyświetlić podsumowanie treningu w postaci tabeli po wciśnięciu przycisku **Result Table**. Przykładowy wynik porównania modeli w postaci tabeli pokazano na rysunku 6.

Session: untitled
Training Data: T Observations: 10000 Predictors: 2 Response Name: Klasa Response Classes: 2
Validation: Holdout validation with 25% held out

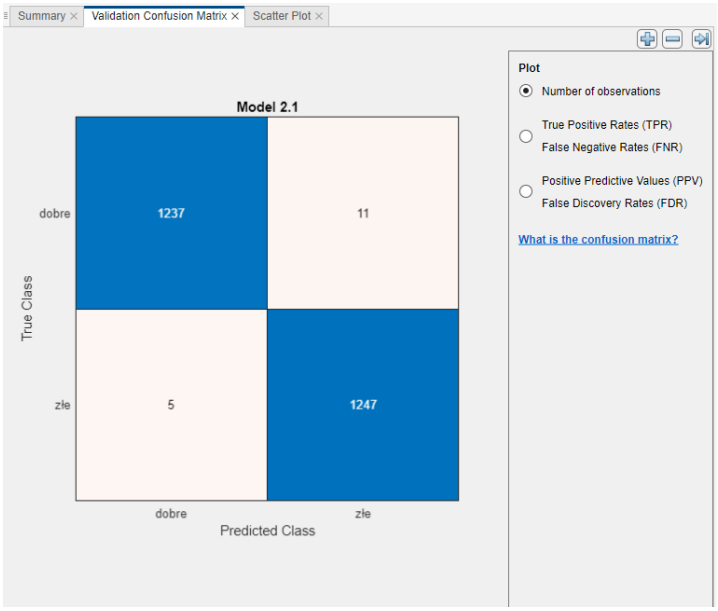
Favorite	Model Num...	Model Type	Status	Accuracy (Validation)	Total Cost (Validation)
☐	2.12	SVM	Trained	100.00 %	0
☐	2.29	Neural Network	Trained	99.96 %	1
☐	2.32	Neural Network	Trained	99.92 %	2
☐	2.30	Neural Network	Trained	99.88 %	3
☐	2.14	SVM	Trained	99.84 %	4
☐	2.15	SVM	Trained	99.80 %	5
☐	2.31	Neural Network	Trained	99.80 %	5
☐	2.28	Neural Network	Trained	99.76 %	6
☐	2.18	KNN	Trained	99.72 %	7
☐	2.21	KNN	Trained	99.72 %	7
☐	2.22	KNN	Trained	99.68 %	8
☐	2.17	KNN	Trained	99.64 %	9
☐	2.24	Ensemble	Trained	99.56 %	11
☐	2.23	Ensemble	Trained	99.48 %	13
☐	1	Tree	Trained	99.36 %	16
☐	2.1	Tree	Trained	99.36 %	16
☐	2.19	KNN	Trained	99.20 %	20

Rys. 6 Widok tabeli Result Table

Wybierając konkretny model w centralnym oknie można sprawdzić parametry i zachowanie się modelu. W zakładce Summary dostajemy podstawowe informacje o modelu, uzyskanej dokładności, czasie treningu oraz rozmiarze całego modelu. Dodatkowo można podejrzeć skuteczność modelu za pomocą macierzy walidacyjnej (Rys. 7 b), gdzie widoczne są liczby poprawnych i negatywnych prób identyfikacji.



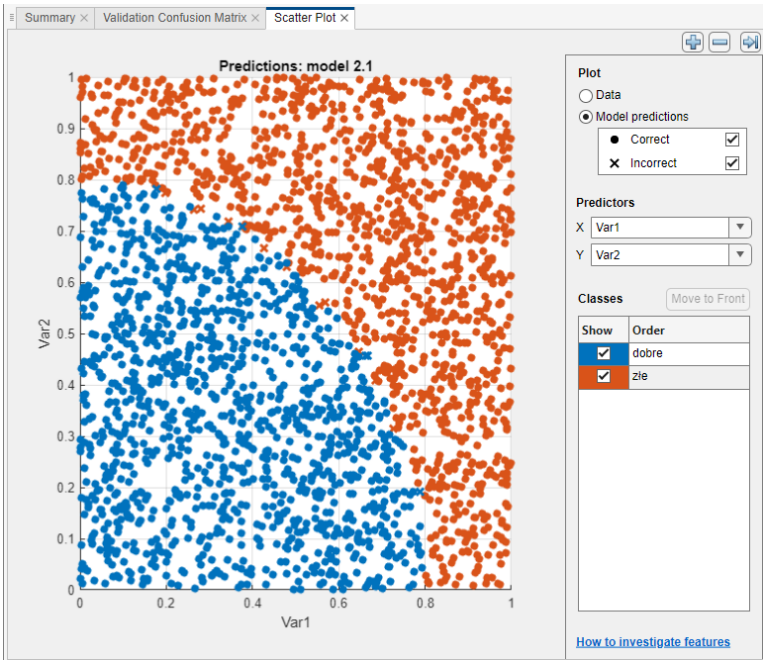
(a)



(b)

Rys. 7 a) Przykładowe okno podsumowania modelu, b) Widok macierzy walidacyjnej

Wejściowe dane uczące i weryfikacyjne można również podejrzeć graficznie w oknie Scatter Plot (Rys. 8). W oknie tym można graficznie podejrzeć/przedstawić na wykresie 2D dane wejściowe i skuteczność wytrenowanego modelu. Scatter Plot jest szczególnie użyteczny w przypadku danych wejściowych o relatywnie małej liczbie wymiarów, gdyż pojedynczy wykres pozwala wykreślić dane dla 2 wybranych zmiennych wejściowych.

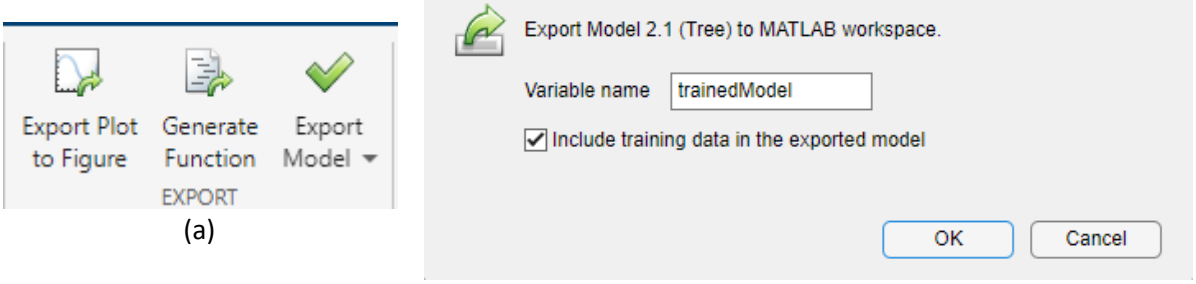


Rys. 8 Widok Scatter Plot – graficzne przedstawienie danych uczących i skuteczności modelu

Na wykresie Scatter Plot dane są kolorowane w zależności od kategorii. Poprawnie zidentyfikowane dane są oznaczane przez kropki, a błędne predykcje oznaczane są krzyżykami (rys.8).

Po przetrenowaniu modeli i sprawdzeniu ich skuteczności można wykorzystać modele w procesie identyfikacji innych danych wykraczających poza dane treningowe. Aby użyć modelu wytrenowanego w **Classification Learner** do klasyfikacji nowych (zewnętrznych) danych, trzeba przejść przez kilka jasno zdefiniowanych kroków.

Najpierw w samej aplikacji należy **wyeksportować model**. Po zakończeniu trenowania należy wybrać najlepszy model i użyć funkcji *Export Model* (Rys. 9 a). MATLAB tworzy wtedy zmienną w Workspace, najczęściej jako strukturę. Przy eksporcie można wybrać nazwę struktury (np. **trainedModel –** Rys. 9b). Struktura modelu zawiera w sobie m.in. funkcje **predictFCN**. Klasyfikacja nowych danych odbywa się właśnie przez tę funkcję.



(a) (b)

Rys. 9 a) Widok przycisków do eksportu danych, b)okna eksportu modelu

Wywołanie funkcji klasyfikacyjnej odbywa się przez komendę:

y_pred = trainedModel.predictFcn(Xnew);

Nowe dane Xnew muszą mieć **dokładnie taką samą strukturę jak dane treningowe**, czyli:

- te same zmienne (kolumny),
- te same nazwy,
- ten sam typ (np. table),
- brak zmiennej decyzyjnej (klasy).

Przykład Dydaktyczny

Aby zobrazować w jaki sposób należy przygotować dane przygotowano skrypt Matlab'a widoczny poniżej. Skrypt służy do przygotowania danych do trenowania modeli dla losowych zmiennych x1 i x2.

Model ma sprawdzać czy współrzędne punktów znajdują się w okręgu o średnicy R. Dane znajdujące się w okręgu oznaczone są jako „dobre" a dane z poza okręgu oznaczono jako „złe". Prezentowany skrypt automatycznie uruchamia narzędzie Classyfication Learner.

```
%% Przykład dydaktyczny rozpoznawania czy dane wejściowe
% leżą wewnątrz koła o promieniu R
% Dane posłużą do testowania narzędzia Classyfication learner

    N=1e4;  % liczba próbek wejściowych
    x1=rand(N,1);
    x2=rand(N,1);   % wygeneruj  próbki wejsciowe
    R=0.8;          % promień koła dla decyzji
    mask = sqrt(x1.^2 + x2.^2) < R;
    Klasa = categorical(mask, [true false], {'dobre','złe'});

% Złozenie tabeli z danymi wejściowymi
    T = array2table([x1,x2]);
% Dodanie kolumny z klasą
    T.Klasa = categorical(Klasa);
% Uruchomienie Classification Learner
    classificationLearner
```

Po uruchomieniu skryptu można w narzędziu Classification Learner rozpocząć nową sesję. Wczytać dane ze zmiennej T z workspace. Wybrać modele do uczenia i uruchomić trening zgodnie z ogólnym opisem.

Deep Network Designer

Deep Network Designer to graficzne środowisko w MATLAB-ie (część *Deep Learning Toolbox*), przeznaczone do projektowania, modyfikacji, trenowania i analizy sieci neuronowych typu deep learning. Jest to narzędzie bardziej zaawansowane niż Classification Learner, ponieważ pozwala pracować bezpośrednio na architekturze sieci neuronowych.

Podstawową funkcją jest możliwość **wizualnego budowania sieci neuronowej** poprzez dodawanie i łączenie warstw, takich jak:

- warstwy konwolucyjne (CNN),
- warstwy pooling,
- warstwy fully connected,
- funkcje aktywacji (ReLU, softmax),
- warstwy normalizacji.

Użytkownik może tworzyć sieć „od zera" lub skorzystać z modeli pretrained (np. ResNet, SqueezeNet) i dostosować je do własnego problemu (transfer learning). Edytor pokazuje strukturę sieci w formie grafu, co ułatwia zrozumienie przepływu danych i zależności między warstwami.

Deep Network Designer (DND) można uruchomić klikając w ikone umieszczoną w **APPS/ Machine Learning and Deep Learning** (Rys. 1) lub wpisując komendę w workspace:

deepNetworkDesigner

W przypadku gdy użytkownik chce wykorzystać już zbudowaną i zapisana sieć np. jako **net**, można ją otworzyć bezpośrednio używając komendy:

deepNetworkDesigner(net)

Etapy pracy w Deep Network Designer

Pracę w Deep Network Designer można ująć w siedmiu logicznych krokach odpowiadających pełnemu pipeline'owi uczenia sieci:

1. Wybór lub utworzenie architektury sieci
Można:

- zbudować sieć od podstaw (dodając warstwy),
- albo wczytać model pretrained (transfer learning).
  Struktura sieci jest przedstawiona graficznie jako graf warstw.

Uwaga! Warstwa wejściowa (Input) musi być dostosowana do danych wejściowych.

2. Przygotowanie i wczytanie danych
Dane (np. obrazy lub sygnały) są organizowane i wczytywane do MATLAB-a. Na tym etapie często wykonuje się wstępne przetwarzanie, np. skalowanie czy podział na zbiory treningowe i walidacyjne.

3. Modyfikacja sieci
Dostosowanie architektury do problemu:

- zmiana liczby neuronów w warstwie wyjściowej,
- dodanie/usunięcie warstw,
- dopasowanie funkcji aktywacji.

4. Konfiguracja procesu uczenia
Ustawienie parametrów treningu:

- optimizer (np. Adam),
- learning rate,
- liczba epok,
- batch size,
- sposób walidacji.

5. Trenowanie sieci
Uruchomienie procesu uczenia z bieżącą obserwacją:

- funkcji straty (loss),
- dokładności (accuracy),
- przebiegu walidacji.

6. Analiza i ocena modelu
Sprawdzenie wyników:

- macierz pomyłek,
- błędy klasyfikacji,
- aktywacje warstw (opcjonalnie).

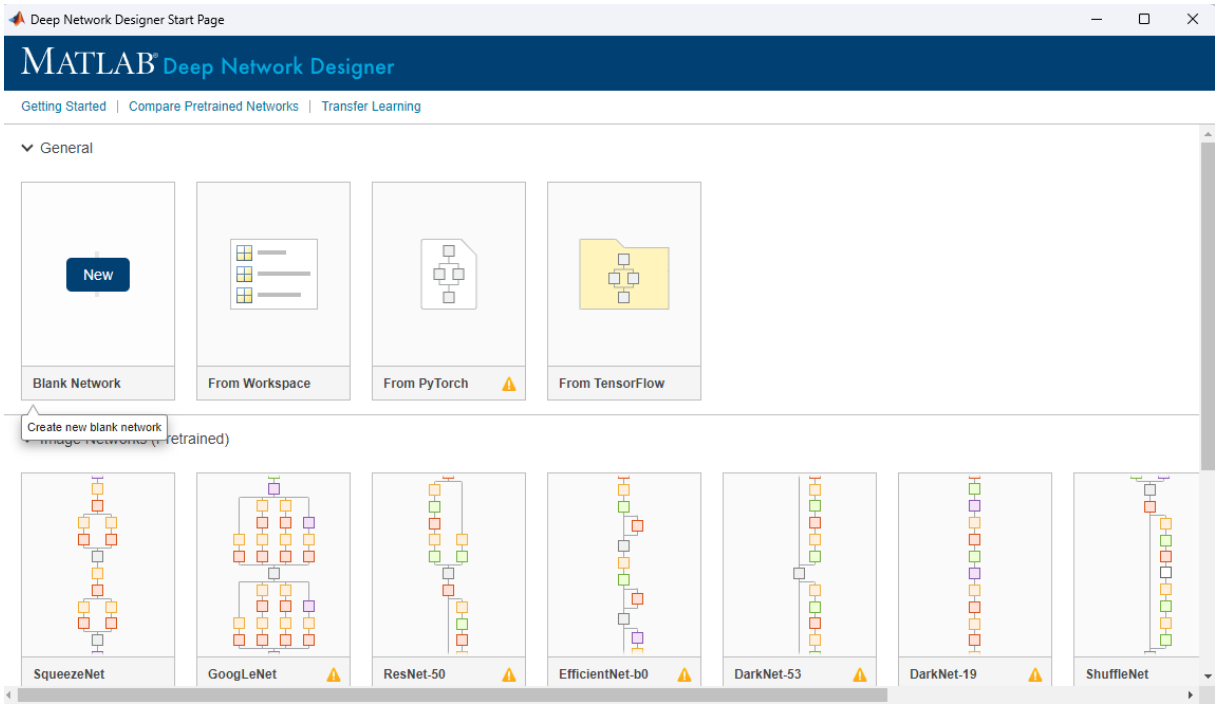
7. Eksport i wykorzystanie modelu
Wytrenowaną sieć można:

- zapisać w Workspace,
- wygenerować kod,

- wykorzystać do klasyfikacji nowych danych lub w Simulinku.

Wybór lub utworzenie architektury sieci

Po uruchomieniu aplikacji DND otwiera się okna startowe (Rys. 10), gdzie można uruchomić nową sesję lub wczytać dostępną siec. Użytkownik ma realnie możliwość zbudowania własnej sieci od podstaw lub wczytanie gotowej sieci. Aby przygotować sieć samodzielnie należy wybrać opcję Blank Network. Poza tym można wczytać przygotowana wcześniej sieć z workspace (opcja From Workspace) lub wczytanie sieci z PyTorch lub Tensor Flow.



Rys. 10 Okno startowe aplikacji Deep Network Designer

TensorFlow to otwartoźródłowa biblioteka programistyczna rozwijana przez Google, służąca do tworzenia i trenowania modeli sztucznej inteligencji, w szczególności sieci neuronowych (deep learning).

Nazwa pochodzi od dwóch pojęć:

- tensor – wielowymiarowa struktura danych (uogólnienie wektora i macierzy),
- flow – przepływ danych w grafie obliczeniowym.

W praktyce TensorFlow umożliwia definiowanie modeli jako **grafów obliczeniowych**, gdzie węzły reprezentują operacje matematyczne, a krawędzie przenoszą dane (tensory).

PyTorch to otwartoźródłowa biblioteka do uczenia maszynowego i deep learningu rozwijana przez Meta Platforms (dawniej Facebook). Jest obecnie jednym z najpopularniejszych narzędzi do budowy i trenowania sieci neuronowych, szczególnie w środowisku badawczym.

Podstawą działania PyTorch są **tensory** (podobnie jak w TensorFlow), czyli wielowymiarowe tablice danych. Kluczową cechą tej biblioteki jest tzw. **dynamiczny graf obliczeniowy (define-by-run)**, co oznacza, że struktura modelu jest tworzona na bieżąco podczas wykonywania kodu.

W aplikacji Deep Network Designer można również wybrać sieć wstępnie wyuczoną **(pretrained networks)**, z listy dostępnych sieci, które wyświetlają się w dolnej części okna startowego (Rys. 10).

Są one wytrenowane na dużych zbiorach (np. ImageNet) i wykorzystywane w transfer learningu. Najważniejsze z nich to:

SqueezeNet
Bardzo lekka i szybka sieć CNN o małej liczbie parametrów. Dobra do zastosowań embedded i szybkiego prototypowania. Sieć ta stanowi kompromis między dokładnością a rozmiarem.

AlexNet
Jedna z pierwszych głębokich sieci CNN. Prosta architektura, ale mniej efektywna niż nowsze modele.

VGG-16 / VGG-19
Głębokie sieci o regularnej strukturze (powtarzalne bloki konwolucyjne). Dobra jakość klasyfikacji, ale duża liczba parametrów i wysokie wymagania obliczeniowe.

GoogLeNet (Inception)
Wprowadza moduły Inception, które równolegle analizują dane różnymi filtrami. Lepsza efektywność niż VGG przy mniejszej liczbie parametrów.

ResNet-18 / ResNet-50 / ResNet-101
Sieci z połączeniami rezydualnymi (skip connections), które rozwiązują problem zanikania gradientu. Bardzo popularne, dobre wyniki i stabilne uczenie.

DenseNet-201
Rozwinięcie idei ResNet – każda warstwa połączona z kolejnymi. Bardzo efektywne wykorzystanie informacji, ale większa złożoność.

MobileNet-v2
Sieć zoptymalizowana pod kątem urządzeń mobilnych. Niskie zużycie zasobów przy zachowaniu dobrej dokładności.

W dalszej części opracowania pokazane zostaną dwa przykłady wykorzystania narzędzia DND do rozpoznawania obrazów. W pierwszym przykładzie pokazane zostanie uczenie transferowe z wykorzystaniem pretreined network. W drugim przykładzie zastosowana będzie sieć budowana w aplikacji od podstaw.

Przykład 1 Uczenie transferowe

Uczenie transferowe (transfer learning) to proces polegający na wykorzystaniu wstępnie wytrenowanej sieci głębokiego uczenia i jej dostrojeniu (fine-tuning) do realizacji nowego zadania. Zastosowanie transfer learningu jest zazwyczaj szybsze i łatwiejsze niż trenowanie sieci od podstaw. Pozwala ono szybko przenieść wyuczone cechy do nowego zadania, wykorzystując mniejszą ilość danych treningowych.

Aby pokazać uczenie transferowe skorzystano z gotowego przykładu opracowanego przez firmę Mathworks – producenta oprogramowania Matlab.

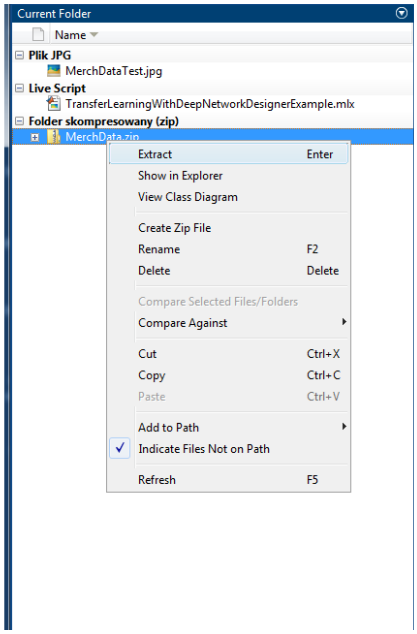
Opis przykładu można znaleźć pod linkiem:

https://www.mathworks.com/help/deeplearning/ug/transfer-learning-with-deep-network-designer.html

Przykład można również uruchomić w programie Matlab wpisując komendę:

openExample("nnet/TransferLearningWithDeepNetworkDesignerExample")

W tym przykładzie firma MathWorks przygotowała zestaw danych uczących i pełny opis kroków postepowania. Przed wykonaniem kolejnych kroków konieczne jest rozpakowanie danych uczących znajdujących się w pliku MerchData.zip. Można to zrobić z poziomu Matlaba klikając prawym przyciskiem myszy na pliku i wybierając opcję Extract (Rys. 11) lub wpisując komendę : **unzip("MerchData.zip"),** lub bezpośrednio rozpakowując ten plik z poziomu eksploratora plików.

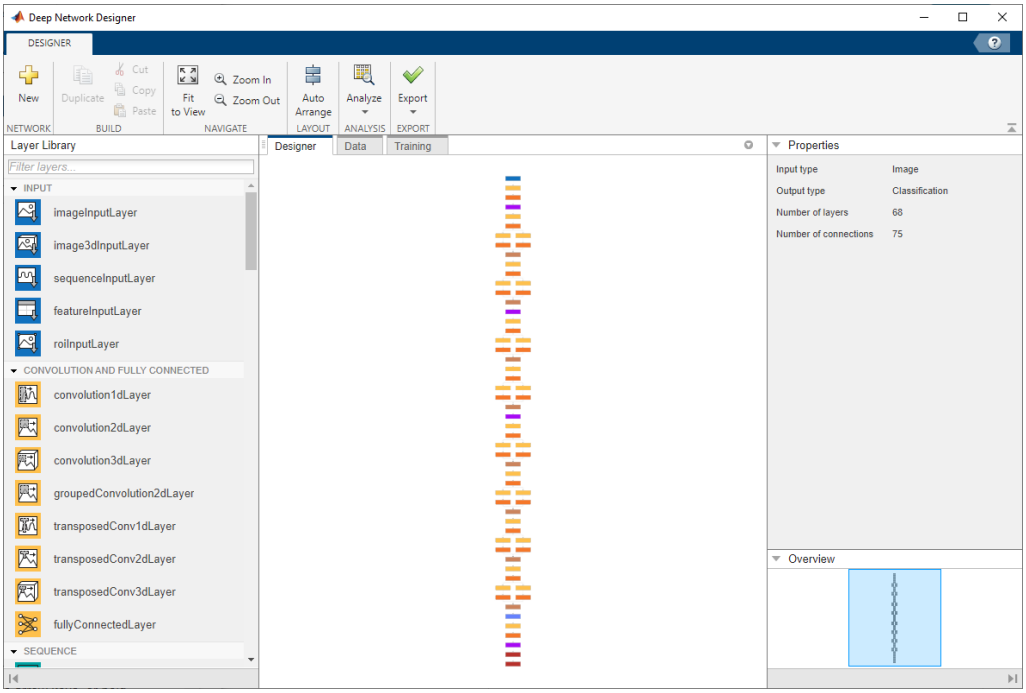


Rys. 11 Rozpakowanie pliku z poziomu Matlab

Aby przeprowadzić transfer learning w klasyfikacji obrazów przy użyciu Deep Network Designer, należy wykonać następujące kroki:

1 **Otwórzyć aplikację Deep Network Designer i wybierać wstępnie wytrenowaną sieć.**

Widok okna wyboru pokazano na rysunku 10. Wybierając pierwszą dostępną sieć SqueezeNet dostaje się widok pokazany na rysunku 12.



Rys. 12 Widok okna Deep network Designer z wybraną siecią.

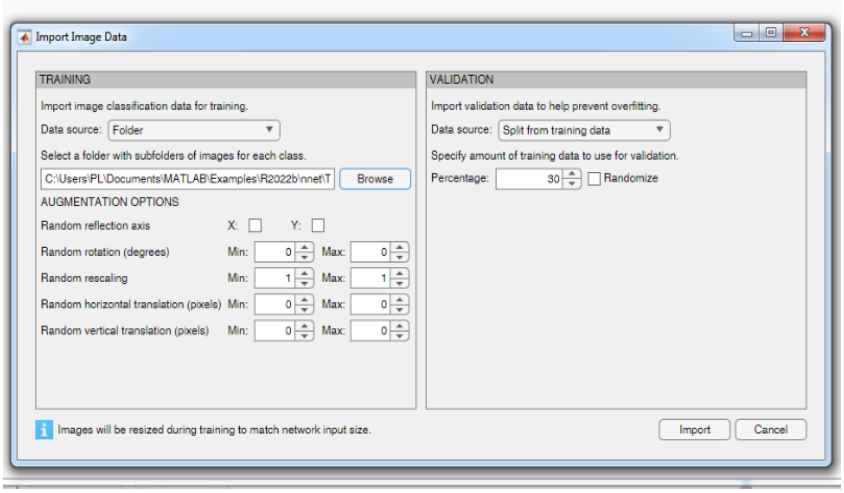
Po lewej stronie okna znajduje się biblioteka elementów sieci z której można wybierać, niczym klocki, elementy do budowy sieci neuronowych. W centralnym oknie dostępne są trzy zakładki

- Designer – gdzie buduje się sieć
- Data – gdzie wybiera i ustawia się parametry danych uczących
- Training – gdzie dobiera się parametry treningu.

Ponieważ wykorzystuje się gotową sieć nie ma konieczności dodawania nowych elementów (choć jest to możliwe i w następnych krokach zostaną zamienione niektóre elementy). W tym momencie można jednak przejść do kolejnego punktu.

2 Import nowego zbioru danych

Dane niezbędne do treningu sieci wybiera się w zakładce Data po kliknięciu w przycisk Import Data. Źródłem danych może być folder z dysku komputera lub dane z workspace. Korzystając z przykładu wystarczy wybrać opcję Folder a następnie wybrać katalog MerchData (Rys.13).



Rys. 13 Widok okna Importu danych uczących.

Ponadto w oknie importu można wybrać opcję AUGMENTATION, która służy do modyfikacji danych treningowych (augmentacja), np.:

- obrót (rotation),
- odbicie (reflection),
- skalowanie (rescaling),
- przesunięcia (translation).

Augmentacja służy do zwiększenia różnorodności danych i poprawy generalizacji modelu.

Finalnie należy także po prawej stronie okna wybrać opcje walidacji modelu – **VALIDATION.**

Można wybrać jedną z 4 opcji dla Data source

(Split from training data) – określa się sposób wydzielenia danych walidacyjnych z danych uczących.

Percentage – procent danych przeznaczonych na walidację (np. 30%).

Randomize – losowy podział danych.

Folder – podaje się oddzielne dane weryfikujące z folderu

None – brak walidacji

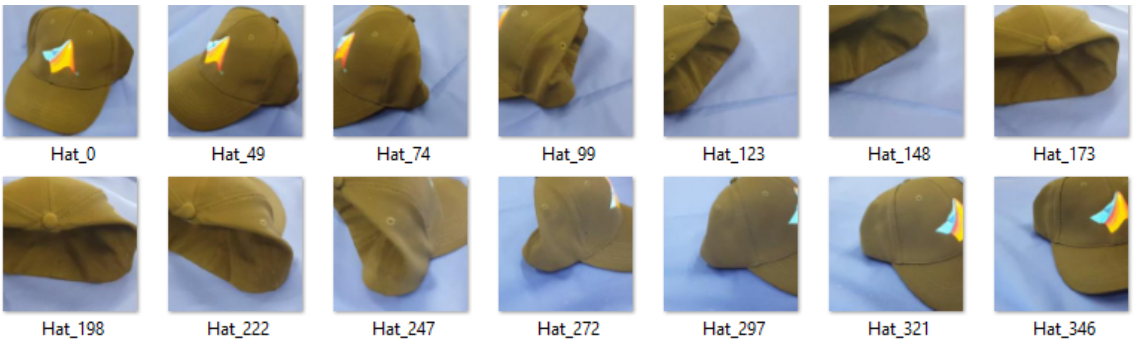
Dane treningowe dostępne w przykładzie składają się z 55 zdjęć i dotyczą 5 obiektów takich jak:

- Talia kart
- -Wkrętak
- -Kostka Rubika
- -Czapka
- -Latarka

Dla każdego z elementów dostępne jest jedynie 11 zdjęć co jest wartością relatywnie małą. Taką liczbę zdjęć można łatwo przygotować dla dowolnych innych obiektów. Niestety ograniczeniem jest

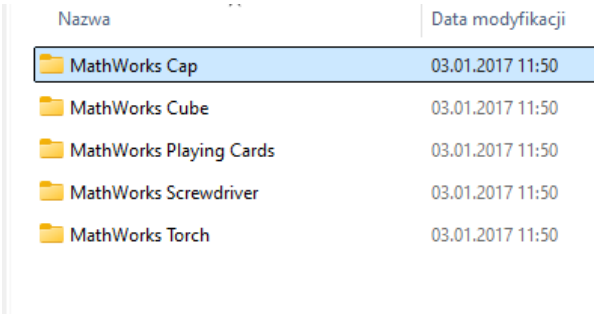
rozdzielczość zdjęć, które muszę być kompatybilne z dostępną siecią. Dla sieci dostępnych z poziomu Deep Network Designer rozdzielczość wynosi 227x227 piksele.

Widok miniatur zdjęć dla czapki i wkrętaka pokazano na rysunku 14.



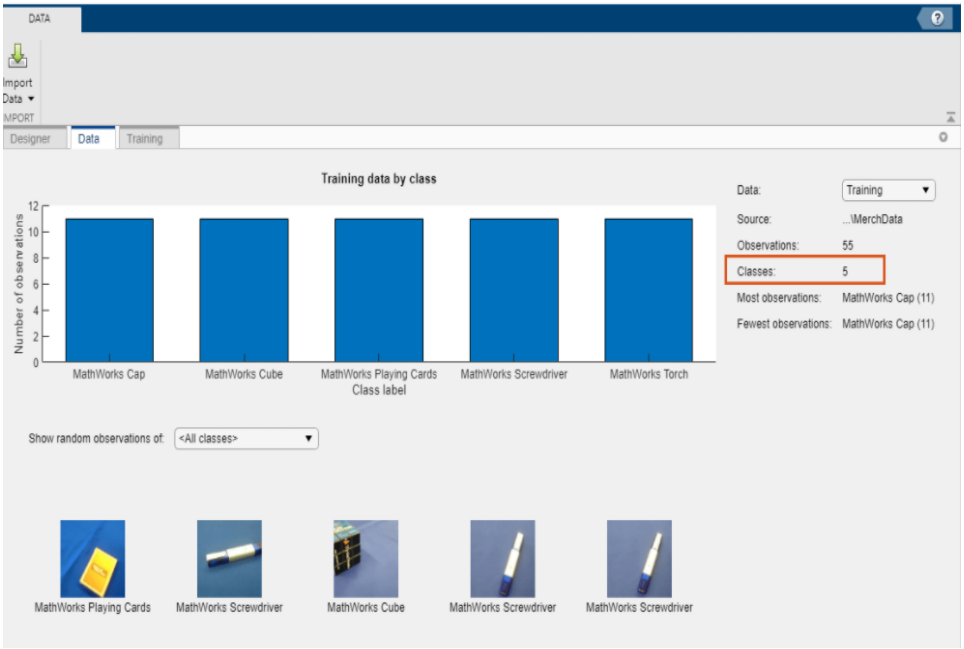
Rys. 14 Widok zdjęć – danych treningowych dla kategorii Czapka- Hat

Istotne jest aby dane treningowe były podzielone i dane dla każdej kategorii znajdowały się w oddzielnym podkatalogu jak pokazano na rys. 15.



Rys. 15 Widok katalogu z podkatalogami zawierającymi dane uczące

Po zaimportowaniu danych można podejrzeć poprawność danych w zakładce data. Można podejrzeć liczbę kategorii i liczność danych dla każdej kategorii. Dodatkowo widok minatur na dole pozwala sprawdzić czy na etapie wyboru danych nie popełniono błądu.



Rys. 16 Widok okna przedstawiającego dane treningowe

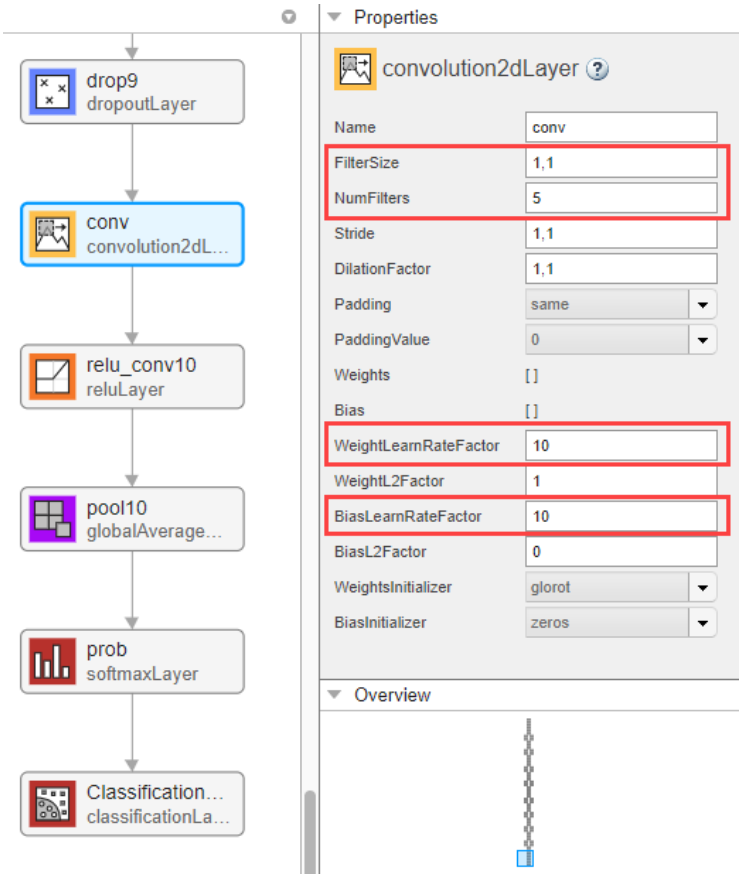
3 Zastąpienie końcowych warstw nowymi warstwami dostosowanymi do nowego zbioru danych.

Po wgraniu danych uczący należy zmienić ostatnie warstwy w modelu. Należy zmienić co najmniej ostatnią warstwę konwolucyjną (convolution2dLayer) i ostatnią warstwę klasyfikacyjną ClasyficationLayer.

Aby to zrobić należy przeciągnąć nową warstwę convolution2dLayer na obszar roboczy (canvas). Aby dopasować ją do oryginalnej warstwy konwolucyjnej, Należy ustawić parametry parametr FilterSize na 1×1 i pozostałe parametry zgodnie z rysunkiem 17. Właściwość NumFilters określa liczbę klas w problemach klasyfikacyjnych. Należy zmienić ją na liczbę klas w nowym zbiorze danych, w tym przykładzie na 5.

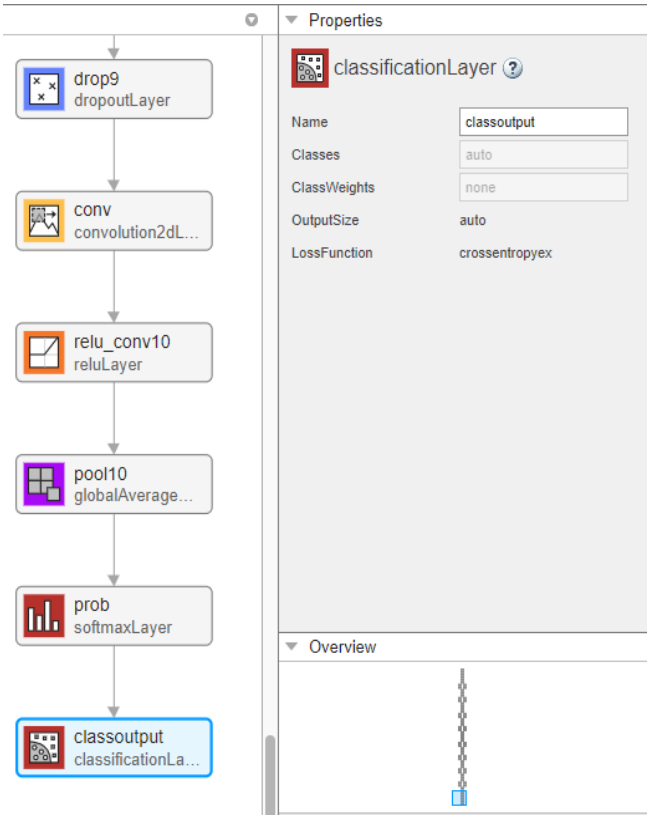
Aby przyspieszyć uczenie w nowej warstwie w porównaniu do warstw przeniesionych, ustawić parametry WeightLearnRateFactor oraz BiasLearnRateFactor na wartość 10.

Finalnie można usunąc ostatnią warstwę konwolucyjną 2-D i w jej miejsce podłącz nowo utworzoną warstwę.



Rys. 17 Zamiana warstwy konwolucyjnej i ustawianie jej parametrów

Analogicznie należy postąpić z ostatnią warstwą ClasyficationLayer, ustawiając ją zgodnie z rysunkiem 18.



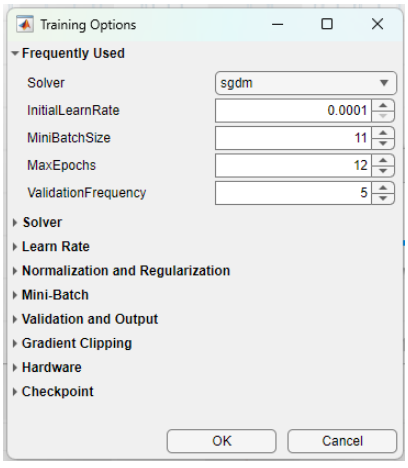
Rys. 18 Zamiana warstwy klasyfikacyjnej i ustawianie jej parametrów

4 Ustawienie współczynników uczenia i wytrenowanie sieci

Ostatnim krokiem w przygotowywaniu modelu sieci neuronowej do klasyfikacji jest wyuczenie modelu. Przed uczeniem można ustawić opcje treningu lub zrobić to z ustawieniami domyślnymi.

Aby wytrenować sieć z domyślnymi ustawieniami, na zakładce Training należy kliknąć Train. Domyślne opcje trenowania są lepiej dostosowane do dużych zbiorów danych; w przypadku małych zbiorów należy zmniejszyć rozmiar mini-batcha oraz częstotliwość walidacji.

Jeśli jest potrzeba większej kontroli nad procesem uczenia, klikamy w Training Options i wybieramy odpowiednie ustawienia (rys. 19)



Rys. 19 Okno ustawień treningu

W celu uzyskania większej kontroli nad procesem uczenia należy wybrać Training Options i skonfigurować odpowiednie ustawienia:

Początkowy współczynnik uczenia (InitialLearnRate) ustawia się na małą wartość, aby spowolnić uczenie w warstwach przeniesionych.

Częstotliwość walidacji (ValidationFrequency) określa się tak, aby dokładność na zbiorze walidacyjnym była obliczana raz na epokę.

Liczbę epok (MaxEpochs) ustawia się na niewielką wartość. Epoka oznacza pełny cykl uczenia na całym zbiorze treningowym. W transfer learningu nie ma potrzeby stosowania dużej liczby epok.

Rozmiar mini-batcha (MiniBatchSize) definiuje liczbę obrazów używanych w jednej iteracji. Aby zapewnić wykorzystanie całego zbioru danych w każdej epoce, należy dobrać mini-batch tak, aby dzielił liczbę próbek treningowych bez reszty.

W analizowanym przykładzie przyjęto:

- InitialLearnRate = 0.0001,
- ValidationFrequency = 5,
- MaxEpochs = 8.

Ponieważ zbiór zawiera 55 obserwacji, ustawiono MiniBatchSize = 11, co zapewnia równomierny podział danych i wykorzystanie całego zbioru treningowego w każdej epoce.

Dodatkowo można wypróbować różne solvery. Solver (Metoda optymalizacji)

- Wybrana opcja: sgdm (Stochastic Gradient Descent with Momentum)

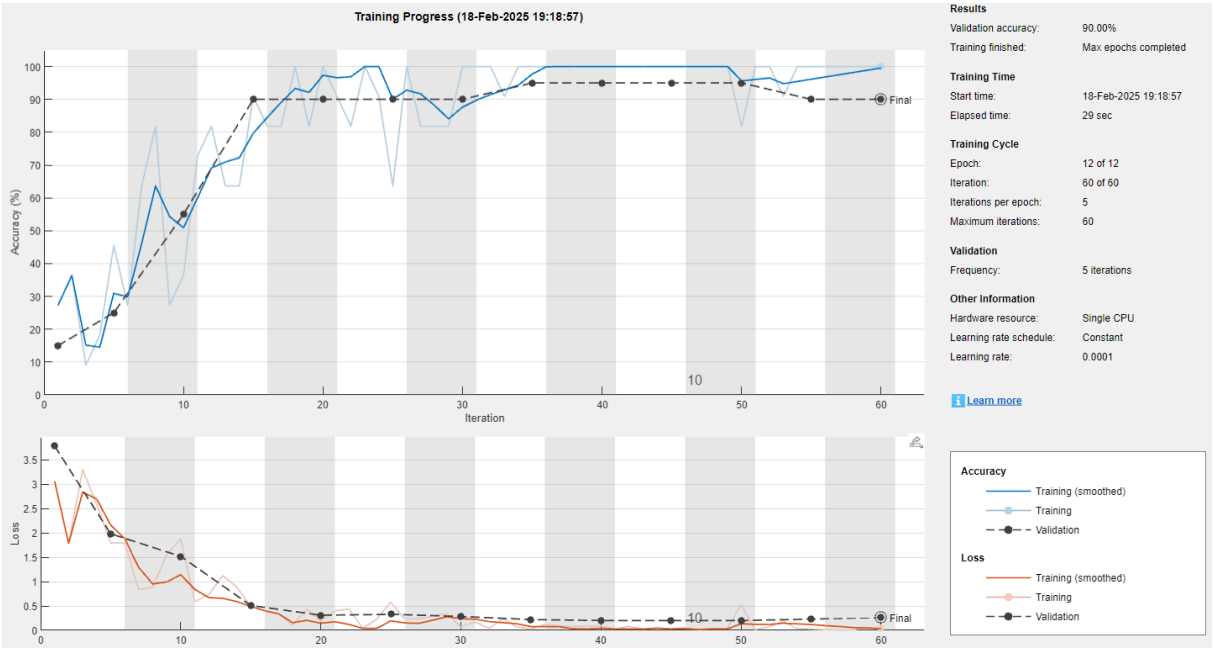
Jest to stochastyczny spadek gradientu z momentum, który pomaga w szybszym znalezieniu minimum funkcji kosztu.

Alternatywnie można wybrać:

- adam – adaptacyjna metoda (często stabilniejsza i lepiej radzi sobie przy małych zbiorach danych).
- rmsprop – dobry dla sieci rekurencyjnych (RNN).

Więcej informacji na temat doboru parametrów trenowania można znaleźć w dokumentacji funkcji trainingOptions.

Po uruchomieniu treningu pojawia się okno (Rys. 20) gdzie na bieżąco można podglądać proces uczenia. W przypadku złych rezultatów można przerwać proces i wrócić do projektowania sieci. Jeśli proces uczeni przebiega poprawnie (z czasem rośnie dokładność) należy poczekać do końca procesu. Finalnie można podejrzeć dokładność modelu



Rys. 20.

5 Wykorzystanie modelu do klasyfikacji

Po zakończonym z sukcesem procesie uczenia można wyeksportować sieć i wykorzystywać ją w procesie klasyfikacji nowych danych.

Aby wyeksportować architekturę sieci wraz z wytrenowanymi wagami, na zakładce Training należy wybrać **Export > Export Trained Network and Results**. Deep Network Designer eksportuje wytrenowaną sieć jako zmienną **trainedNetwork_1**, a informacje o procesie trenowania jako zmienną **trainInfoStruct_1.**

Możliwe jest również wygenerowanie kodu MATLAB, który odtwarza architekturę sieci oraz użyte opcje trenowania. W tym celu na zakładce Training należy wybrać Export > Generate Code for Training. Analiza wygenerowanego kodu pozwala zapoznać się ze sposobem programowego przygotowania danych do trenowania, tworzenia architektury sieci oraz przeprowadzania procesu uczenia.

Następnie w Matlab można pobrać nowe dane do workspace. W tym celu można wykorzystać komendę imread:

I = imread("MerchDataTest.jpg");

Ponieważ, sieć obsługuje jedynie zdjęcia w jednym formacie – wybranym na etapie uczenia, konieczne jest ewentualne skorygowanie rozmiarów zdjęcia, które będzie klasyfikowane. Do tego można użyć funkcji imresize. W naszym przypadku sieć była przystosowana do rozmiaru 227x227 pikseli.

I = imresize(I, [227 227]);

Gdy wczytane jest zdjęcie o odpowiednim rozmiarze można użyć kod Matlab do identyfikacji/klasyfikacji zdjęcia ze zmiennej I:

```
[YPred,probs] = classify(trainedNetwork_1,I);

imshow(I)

label = YPred;

title(string(label) + ", " + num2str(100*max(probs),3) + "%");
```

Uzyskuje się wyniki:

YPred - Jest to przewidziana etykieta klasy (prediction) dla danych wejściowych (może być kilka)

probs - Jest to macierz prawdopodobieństw przynależności do każdej klasy.

Przykład 2 Budowa własnego modelu

Uczenie transferowe pokazane w poprzednim podpunkcie, mimo dużej skuteczności ma swoje ograniczenia. Po pierwsze sieć wstępnie wyuczona obsługuje tylko jedną rozdzielczość, która typowo jest niska i nie pozwala uchwycić wielu szczegółów, które mogą być dla nas istotne. Rozmiar sieci głębokich jest duży i rośnie przy zwiększaniu rozmiarów plików wejściowych. Rozpoznawanie elementów zbliżonych do siebie kształtem i kolorem może być utrudnione lub wręcz niemożliwe.

Chcąc wyeliminować te ograniczenia można budować swoje sieci neuronowe, również głębokie. W takim przypadku większa swoboda w budowie sieci wymaga większej liczby danych uczących i dłuższego procesu uczenia.

Proces postępowania przy budowie własnego modelu pokrywa się w dużej mierze z uczeniem transferowym. Różnicą jest to że na początku wybieramy opcję **New – Blank Network** w oknie z rysunku 10.

Po otwarciu okna aplikacji Deep Network Designer wygodnie jest pobrać dane wejściowe, gdyż późniejsze wprowadzanie warstw modelu spowoduje, że ustawione domyślnie parametry warstw wejścia i wyjścia będą dopasowane do danych.

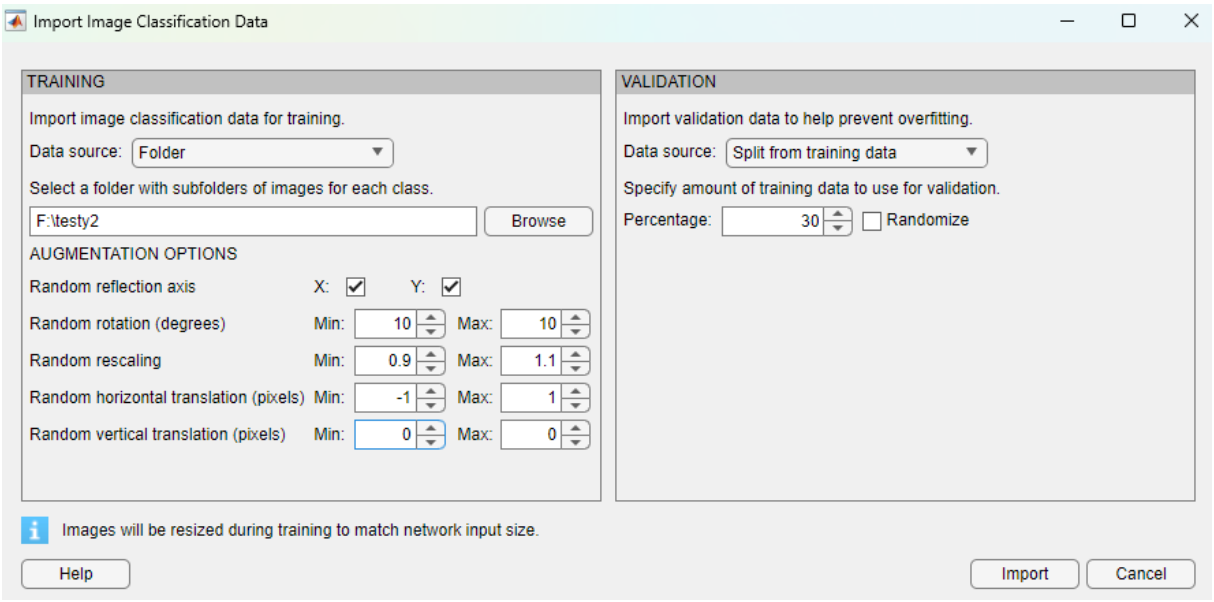
W tym przykładzie przygotowano serię danych po 400 obrazów dla dwóch kategorii.

Na jednych obrazach są koła, a na drugich kwadraty. Obrazy mają po 420 x420 pikseli i dla każdej kategorii przygotowano po 100 obrazów. Obrazy przygotowano w katalogu Dane_treningowe2.

Wczytanie danych wejściowe

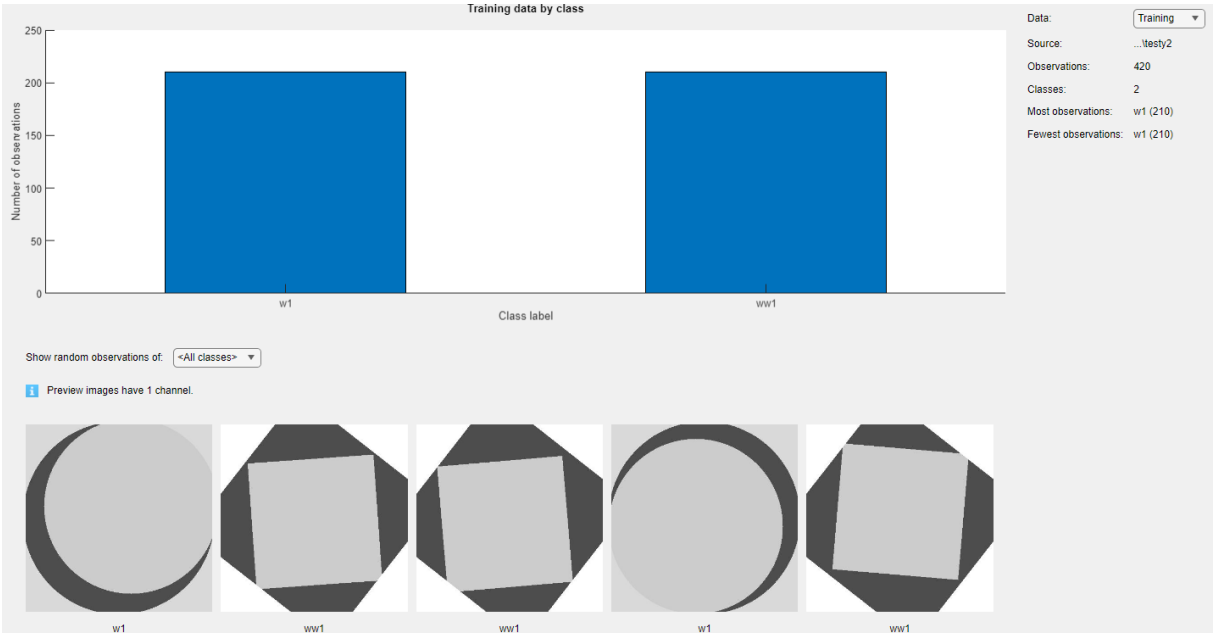
W zakładce Data kliknąć ikonę Import Data i wybrać z dysku katalog Dane_treningowe2.

Wprowadzić opcje Augmentacji zgodnie z rysunkiem 21 i zaakceptować wybór klikając przycisk Import.



Rys. 21 Okno wyboru danych treningowych

Następnie otworzy się okno z podglądem danych treningowych widoczne na rysunku 22. W oknie tym można sprawdzić, czy wczytano poprawnie dane. W szczególności istotne jest sprawdzenie liczby klas oraz liczby danych treningowych. W podglądzie można również zobaczyć, czy wybrano właściwe obrazy i wykluczyć pomyłkę, związaną na przykład z wczytaniem błędnego katalogu.



Rys. 22 Widok okna podglądu danych treningowych

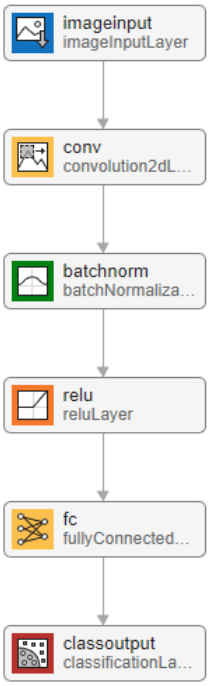
W kolejnym etapie trzeba zbudować strukturę sieci identyfikującej. Strukturę sieci buduje się w oknie Designer. Widok prostej sieci pokazano na rysunku 23. W rozważanym przypadku zbudowano sieć sześciowarstowową. Jednakże narzędzie służy właśnie do eksperymentów dobierania elementów sieci

we własnym zakresie. Budowa sieci jest intuicyjna i polega na wybieraniu elementów sieci z zakładki **Layer Library** i przesuwaniu ich myszą do okna designer i łączeniu ze sobą elementów domyślnymi liniami.

Równoważnie można przygotować sobie strukturę sieci tworząc zmienną matlaba w workspace. Przykład budowy modelu sieci pokazano w ramce poniżej. W przykładzie tworzymy zmienną layers typu LAYER, którą możemy wczytać do aplikacji (ostatnia komenda w ramce).

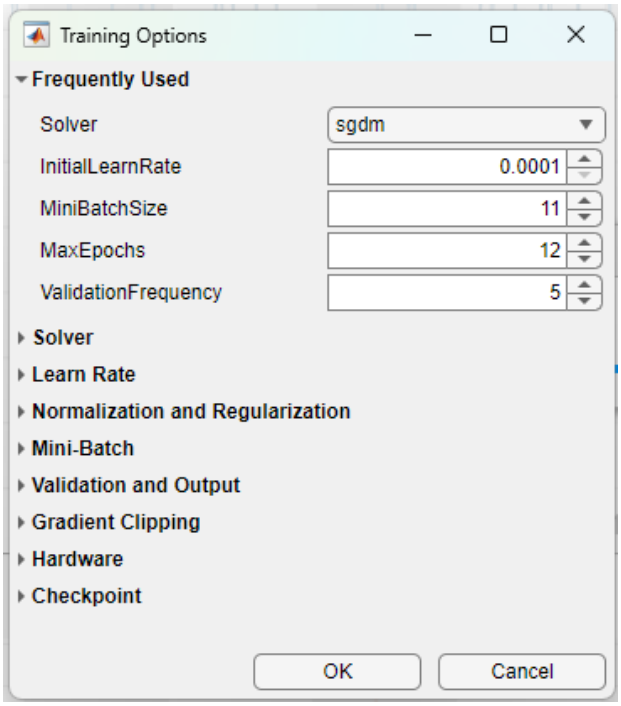
```
layers=[imageInputLayer([420 420 1],"Name","imageinput")
convolution2dLayer(3,32,"Padding","same","Name","conv")
batchNormalizationLayer("Name","batchnorm")
reluLayer("Name","relu")
fullyConnectedLayer(2,"Name","fc")
classificationLayer("Name","classoutput")];

deepNetworkDesigner(layers)
```



Rys. 23 Widok prostej sieci 6 warstwowej

Po przygotowaniu modelu możemy przejść do etapu trenowania sieci (zakładka **Training** w centralnym oknie aplikacji DND). Proces treningu rozpoczynamy od ustawienia opcji treningu (**Training Options**). Okno „Training Options" w MATLAB Deep Network Designer, pokazane na rysunku 24, służy do konfiguracji procesu uczenia sieci neuronowej. To właśnie te parametry w dużej mierze decydują o jakości uzyskanego modelu, często bardziej niż sama architektura sieci.



Rys. 24 Widok okna wyboru opcji treningowych.

Jednym z podstawowych parametrów jest wybór algorytmu optymalizacji (**Solver**). W analizowanym przypadku zastosowano metodę **SGDM** (Stochastic Gradient Descent with Momentum), która jest stabilnym i często stosowanym algorytmem uczenia. Alternatywnie można wykorzystać metodę Adam, która zazwyczaj zapewnia szybszą konwergencję i jest dobrym wyborem w początkowej fazie eksperymentów.

Kolejnym kluczowym parametrem jest początkowy współczynnik uczenia (**Initial Learn Rate**). Określa on krok, z jakim aktualizowane są wagi sieci. Zbyt duża wartość może prowadzić do niestabilności procesu uczenia, natomiast zbyt mała znacząco go spowalnia. Typowe wartości mieszczą się w zakresie od 0.001 do 0.0001. W analizowanej konfiguracji zastosowano wartość 0.0001, co zapewnia stabilność kosztem szybkości uczenia.

Istotnym parametrem jest również rozmiar mini-partii (**MiniBatchSize**), który określa liczbę próbek przetwarzanych jednocześnie. Mniejsze wartości poprawiają zdolność generalizacji modelu, natomiast większe przyspieszają obliczenia, szczególnie przy wykorzystaniu GPU. Typowe wartości mieszczą się w zakresie od 8 do 128. W analizowanym przypadku zastosowano wartość 11, co jest poprawne dla niewielkich zbiorów danych.

Parametr **MaxEpochs** określa liczbę epok, czyli pełnych przejść przez zbiór treningowy. Zbyt mała liczba epok prowadzi do niedouczenia modelu, natomiast zbyt duża może powodować przeuczenie. Wartości rzędu 10–30 stosuje się do wstępnych eksperymentów, natomiast dla właściwego treningu często wykorzystuje się zakres 50–200. W pokazanym na rys. 24 przypadku ustawiono 12 epok.

ValidationFrequency określa, co ile iteracji przeprowadzana jest walidacja modelu. Parametr ten umożliwia monitorowanie jakości uczenia i wykrywanie przeuczenia. Częstsza walidacja daje lepszą kontrolę nad procesem, ale wydłuża czas obliczeń.

Oprócz podstawowych parametrów istotne są również dodatkowe sekcje konfiguracyjne. W szczególności warto zwrócić uwagę na harmonogram zmiany współczynnika uczenia (**Learn Rate Schedule**), który pozwala zmniejszać krok uczenia w trakcie treningu, co często poprawia końcową

jakość modelu. Ważna jest także regularyzacja (np. L2), ograniczająca przeuczenie, oraz mechanizmy walidacyjne, takie jak early stopping, które zatrzymują trening w momencie pogorszenia wyników.

Po skonfigurowaniu procesu uczczenia klikamy w przycisk Train. Powinno pokazać się okno uczenia (rysunek 25) Po zakończonym procesie można ocenić uzyskane efekty.

Rys. 25 Okno pokazujące proces uczenia sieci.

W prezentowanym przykładzie sieć nauczyła się rozpoznawać/odróżniać obrazy z kołami od obrazów z kwadratami ze skutecznością 100%. Można zatem przyjąć, że przyjęta sieć o relatywnie małej liczbie warstw jest odpowiednia do postawionego problemu. W przeciwnym razie należy wrócić do etapu projektowania sieci i przeprowadzić procedurę od początku.