Rozpoznawanie monet

Adrian Matusiak

21 czerwca 2022

1 Założenia projektowe

Pomysł na projekt - notabene mój pierwszy z uczenia maszynowego, ponieważ nie miałem z tym wcześniej do czynienia, został wzięty z jednej z list przykładowych projektów ML dla osób średniozaawansowanych. Chciałem w nim połączyć zdobycie wiedzy związanej ze sztuczną inteligencją ze zdobyciem biegłości w przetwarzaniu obrazów w Pythonie (do tej pory proste rzeczy w MATLAB-ie oraz C/C++, jednak bez OpenCV).

Przeglądając literaturę na ten temat, moją uwagę zwróciły różnego rodzaju algorytmy obliczeniowe przetwarzania obrazów, służące do wykrywania cech, takie jak transformacje Hougha, polarne, zastosowania DFT i FFT, Canny edge - służące do wykrywania krawędzi oraz inne. Do redukcji cech miały zostać wykorzystane transformacje scale-invariant lub algorytmy LDA lub PCA. W projekcie miały wystąpić algorytmy klasyczne, takie jak MLP (w miarę możliwości wspomagane LDA lub PCA), SVM oraz sieci konwolucyjne. Większość projektu miała być wykonana w Kerasie, część prostszych zadań w sklearn.

Pierwotnie projekt zakładał pisanie własnych struktur i klas, które mogłyby być łatwo wizualizowane za pomocą schematów w miarę możliwości zgodnymi z UML. Do tego planowałem stworzyć interfejs, jednak ze względu na złożoność obiektów i ogólny chaos, który wprowadziły niektóre modele, wycofałem się z tego, starając się skupić jak najbardziej na samych algorytmach, ich cechach i tym, jakie wyniki otrzymałem.

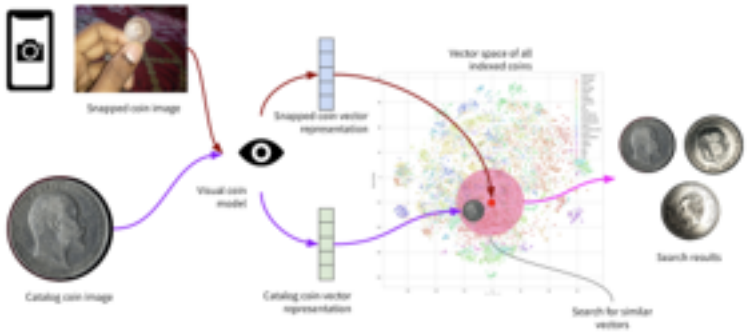
Pierwotne podejście, znane z programowania obiektowego niestety się nie sprawdziło. W sprawozdaniu przedstawiam za to poszczególne kroki modelowania, wykonane przeze mnie po refinemencie konspektu.

2 Przegląd literatury

W ramach przeglądu literatury zostały rozpoznane prace, wykorzystujące algorytmy takie jak SVM, w połączeniu z LDA i FFT (Rysunek 1), gdzie algorytm FFT był wykorzystany jako transformacja ortogonalna do wydzielenia cech niezależnych, a algorytm LDA do ich redukcji. Autor błędnie podaje LDA jako algorytm uczący, czego nie byłem świadomy na tym etapie projektu. Ponadto - projekt jest niestety niedokończony - nie zawiera ocen jakości wytrenowanej sieci.

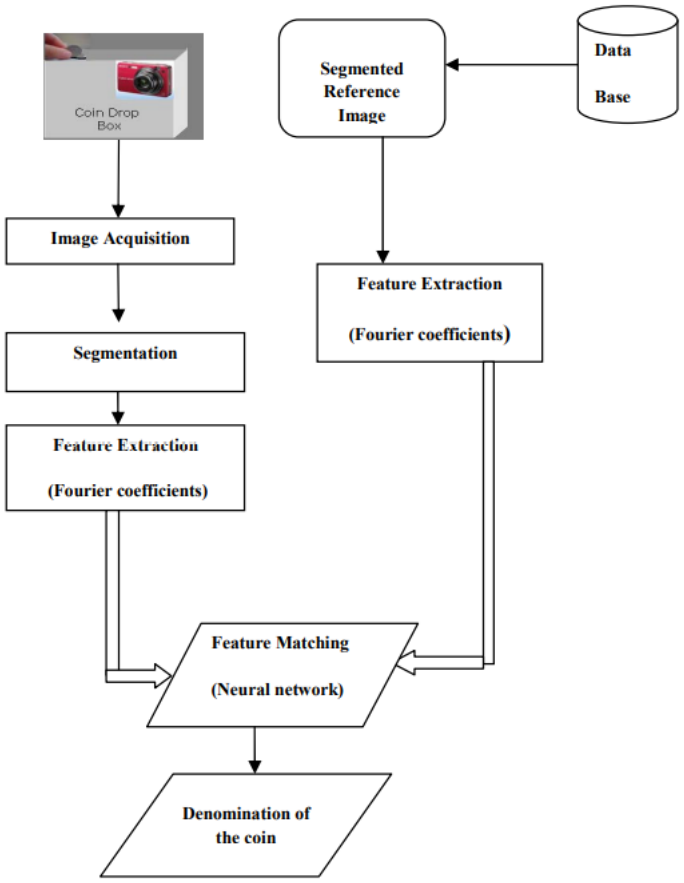
Rysunek 1: Screenshot z githuba jednego z popularniejszych projektów

Kolejnym projektem, który zwrócił moją uwagę był projekt Grid Dynamics - świetnie opisany projekt, jednak podejście znacznie różniło się od klasycznego i nie mogłem go odtworzyć. Ten algorytm wykorzystywał sieci konwolucyjne do uczenia, a do wyszukiwania algorytm k-najbliższych sąsiadów. Schemat przedstawiam na rysunku nr 2.



Rysunek 2: Wizualizacja działania projektu Grid Dynamics

Ponadto natknąłem się na dwie sieci opisane w artykułach z żurnali IEEE. Niestety nie były one pomocne. Z jednej strony pokazały mi, że wybór sieci MLP w połączeniu z PCA lub LDA powinien przynieść odpowiednie wyniki, więc potwierdziło moje założenia, niestety przebieg projektu pokazał, że wykonanie tego nie jest takie proste. Być może to sprawiło, że te pozornie niezbyt ambitne tematy, zostały przestawione na prestiżowych konferencjach. Niemniej, niestety bardziej niż na samym uczeniu maszynowym, skupiają się one na przetwarzaniu obrazów. Przykładowe schematy z przytoczonych artykułów naukowych przedstawiam na poniższych grafikach:



Rysunek 3: Architektura jednego z systemów opublikowanych w IEEE

Ogólny wniosek z researchu jest taki, że algorytmy zostały dobrane dobrze i do dalszego etapu realizacji projektu przeszły algorytmy MLP, SVM i CNN.

3 Dataset

Projekt zaczął się od wybrania datasetu - moją ambicją było wybranie czegoś skomplikowanego, stąd wybrałem dataset World Coins, zawierający w zbiorze treningowym 843 zdjęcia, przydzielone do 211 klas. 211 rodzajów monet pochodzi z 32 walut. Zdjęcia pochodzą ze strony ucoin.net oraz innych źródeł internetowych. Dataset jest niezbalansowany, jednak różnice między ilością zdjęć w poszczególnych klasach nie są duże. Jego zbalansowanie do 28 zdjęć treningowych na klasę zostało wykonane w trakcie budowy jednego z modeli. Dużą zaletą datasetu były wydzielone już wcześniej zbiory testowe i walidacyjne. Dataset miał 38 pozytywnych recenzji i zostało na nim wykonanych 6 różnych

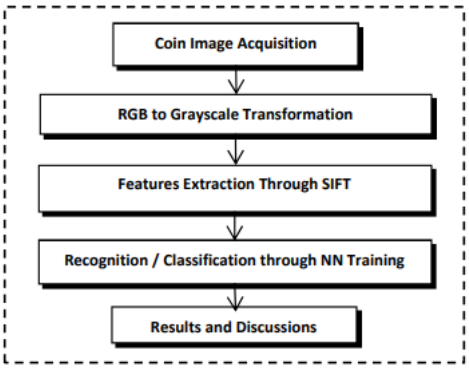


Figure 1. Architecture of coin recognition system

Rysunek 4: Architektura drugiego z systemów z IEEE

```
Found 6413 validated image filenames belonging to 211 classes.        for train generator
Found 844 validated image filenames belonging to 211 classes.         for valid generator
Found 844 validated image filenames belonging to 211 classes.         for test generator
test batch size:  4   test steps:  211  number of classes :  211
```

Rysunek 5: Wynik augmentacji

projektów. Autorem datasetu jest Pablo Lopez Santori, a dataset został ostatni raz zaktualizowany w 2019 roku.

4 Preprocessing

Jedną z pierwszych rzeczy, jakie postanowiłem zrobić, było zbalansowanie datasetu oraz stworzenie nowych obrazów na podstawie posiadanego zbioru. Augmentacja polega na wykonywaniu lustrzanych odbić, obrotów itp. Wykorzystany został do tego ImageGenerator z Kerasa.

Code Listing 1: ImageGenerator

```
trgen=ImageDataGenerator(horizontal_flip=True,rotation_range=20, width_shift_range
    =.2,height_shift_range=.2, zoom_range=.2 )
t_and_v_gen=ImageDataGenerator()
```

Output był następujący: Przykładowe zdjęcia przed i po augmentacji pokazuję na przykładzie złotówek (co ciekawe, w zbiorze są też złotówki z czasów PRL!)

5 Algorytm MLP

Pierwszym zbudowanym modelem klasycznym był model MLP. Składał się on z trzech warstw, zbudowanych sekwencyjnie w Kerasie. Wykonane były

Rysunek 6: Zdjęcia złotówek przed augmentacją

próby kompresji obrazu do rozmiaru 256x256 pikseli, zakończone niepowodzeniem. W związku z tym, postanowiłem pójść dalej i skompresować obraz do 25x25 pikseli. To również nie przyniosło rezultatu. Przedstawiam fragment kodu z budowy modelu i jeden z outputów.

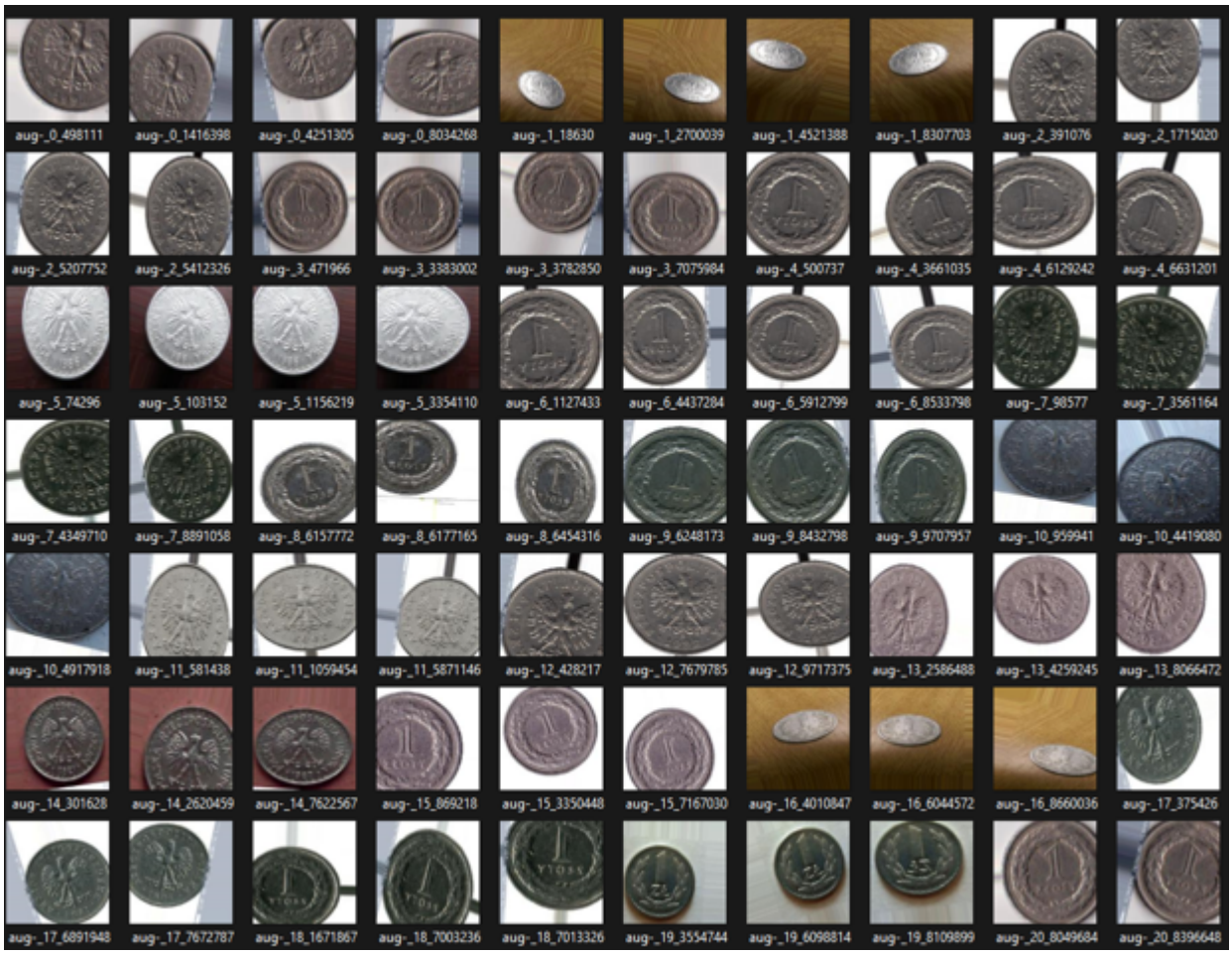
```
model = Sequential([
    Flatten(input_shape=(32,256,256,3)),

    # dense layer 2
    Dense(128, activation='sigmoid'),

    # output layer
    Dense(3, activation='sigmoid'),
])
#%%
model.compile(optimizer='adadelta', loss='categorical_crossentropy', metrics=['accuracy'])
[print(i.shape, i.dtype) for i in model.inputs]
[print(o.shape, o.dtype) for o in model.outputs]
[print(l.name, l.input_shape, l.dtype) for l in model.layers]
#%%
model.fit(X_train, y_train, epochs=40, batch_size=2000)
```

Dziś wiem, że popełniłem dwa błędy - wybrałem zły optimizer oraz przede wszystkim - kompresja z obrazów mających ponad 500 pikseli do 25 pikseli to zdecydowanie za dużo. Z kolei sieć dla 256 pikseli zawierała błędy w strukturze, a tego algorytm MLP nie wybacza. Stąd przykładowe printy w kodzie, które stosowałem do weryfikacji, jakiego kształtu i typu powinny być dane. Przestawiałem to dokładniej na prezentacji. Podsumowując:

- LP nie rozumie „obiektów 2D”, wymagany jest reshape,
- Pixel = feature, np. dla klasycznych zbiorów CIFAR100 możemy zrobić reshape to 25x25 = 225 features, stąd obrazy są problematyczne,



Rysunek 7: Zdjęcia złotówek po augmentacji

```
Training will proceed until epoch 5  then you will be asked to
 enter H to halt training or enter an integer for how many more epochs to run then be asked agai
Epoch 1/40
214/214 [==============================] - 55s 256ms/step - loss: 11.0614 - accuracy: 0.0064 -
val_loss: 10.7218 - val_accuracy: 0.0047
Epoch 2/40
214/214 [==============================] - 49s 227ms/step - loss: 11.0544 - accuracy: 0.0064 -
val_loss: 10.7696 - val_accuracy: 0.0047
Epoch 3/40
214/214 [==============================] - ETA: 0s - loss: 11.0531 - accuracy: 0.0069
Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.
214/214 [==============================] - 47s 222ms/step - loss: 11.0531 - accuracy: 0.0069 -
val_loss: 10.7702 - val_accuracy: 0.0047
Epoch 4/40
214/214 [==============================] - 53s 247ms/step - loss: 11.0491 - accuracy: 0.0076 -
val_loss: 10.7703 - val_accuracy: 0.0047
Epoch 5/40
214/214 [==============================] - ETA: 0s - loss: 11.0487 - accuracy: 0.0072
Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0024999999441206455.
Restoring model weights from the end of the best epoch.
214/214 [==============================] - 53s 246ms/step - loss: 11.0487 - accuracy: 0.0072 -
val_loss: 10.7698 - val_accuracy: 0.0047
Epoch 00005: early stopping
training elapsed time was 0.0 hours,  4.0 minutes, 18.40 seconds)
```

Rysunek 8: Próba wytrenowania sieci zakończona niepowodzeniem

- Musimy zadbać o to, aby przynajmniej pierwszy wymiar był taki sam i dopasować modele do wejść – kluczowy jest tutaj kształt
- Wyniki – Poniżej 40 iteracji „nie działa". A czasem i powyżej. Wymaga wysokiego wytrenowania sieci,
- MLP nie jest fault-tolerant. Drobny błąd przy konstrukcji modelu sprawi, że po prostu się nie uda.
- Mimo bycia jednym z pierwszych i bardziej klasycznych modeli, dla obrazów jest trudny do zbudowania - co innego dla danych zaklasyfikowanych w pliku csv
- Co ciekdawe - dalej rozwijany, zwłaszcza z algorytmami LDA i PCA do nowych zastosowań, ale też w mieszanych modelach - Google stworzyło Mixed-MLP na przykład.

6 Algorytm SVM

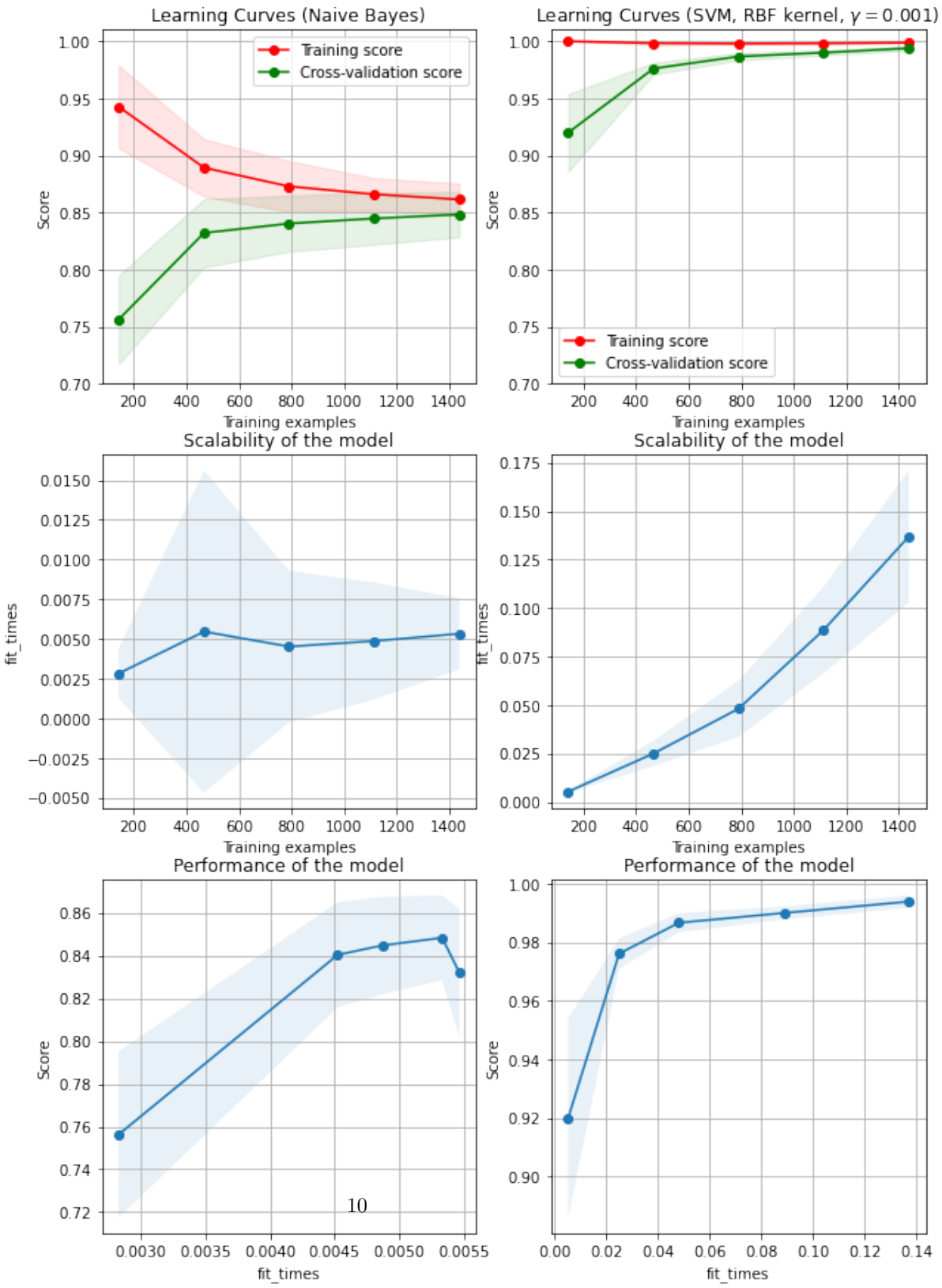
Algorytm SVM był parametryzowany dwoma kernelami rbf i poly. Lepsze wyniki dawał RBF. Do strojenia wykorzystany został GridSearch, który okazał się bardzo czasochłonny. Predykcja wykonania tego algorytmu dla wszystkich 211 klas pozwala z dużym prawdopodobieństwem zakładać, że osiągnąłby dokładność powyżej 0.9. Ogólna charakterystyka mojego modelu SVM:

- Bardzo mało ręcznego strojenia parametrów - tylko wybór jądra
- Skuteczne modele, zwłaszcza dla większej ilości danych
- Brak większych problemów z preprocessingiem danych, brak nawet większej potrzeby ekstrakcji cech przy pomocy algorytmów
- Bardzo długo się uczy wraz z GridSearchem: 4 klasy – 8 minut, 18 klas – 65 minut, 211 klas… postanowiłem już tego nie sprawdzać.
- Częściowo klasyczny algorytm, który w niektórych przypadkach potrafi dorównać lub przewyższyć CNN Wyniki: 0.64 dla 4 klas, 0.74 accuracy dla 18 klas.

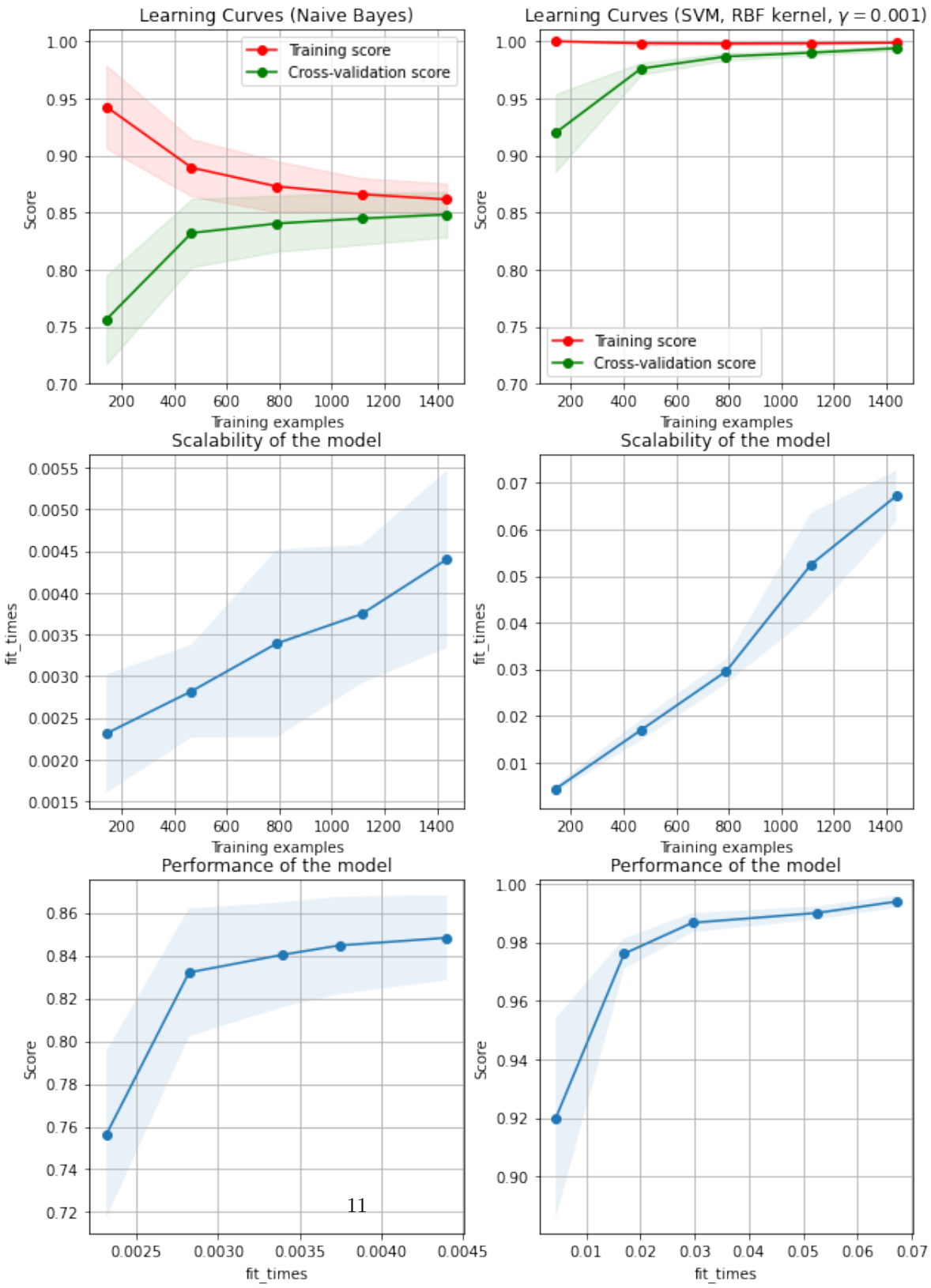
Poniżej przedstawiam krzywe uczenia:

7 Algorytm CNN - EfficientNetB3

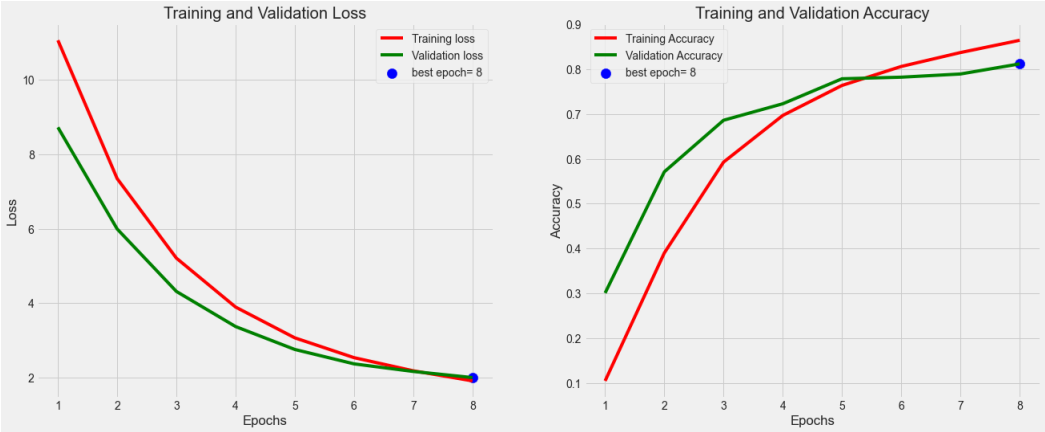
Został tutaj odtworzony najlepiej oceniany projekt z Kaggle. Wykonałem uczenie krok po kroku wg poradnika w swoim środowisku. Sieć bazuje na modelu EfficientNetB3, który sam sobie wykrywa cechy. Wykonano 8 generacji, czas uczenia wyniósł 2h24min, dopasowanie na pełnym zbiorze to 0.83. Zaletą takiego podejścia błyskawiczność adaptacji rozwiązania z internetu - i jego skuteczność. W tym przypadku przedstawiam krzywe walidacyjne poszczególnych generacji i raport z klasyfikacji.



Rysunek 9: Krzywa uczenia dla 5 kategorii



Rysunek 10: Krzywa uczenia dla 18 kategorii



Rysunek 11: Zastosowanie gotowej sieci CNN

8 Sieć CNN połączona z warstwami MLP

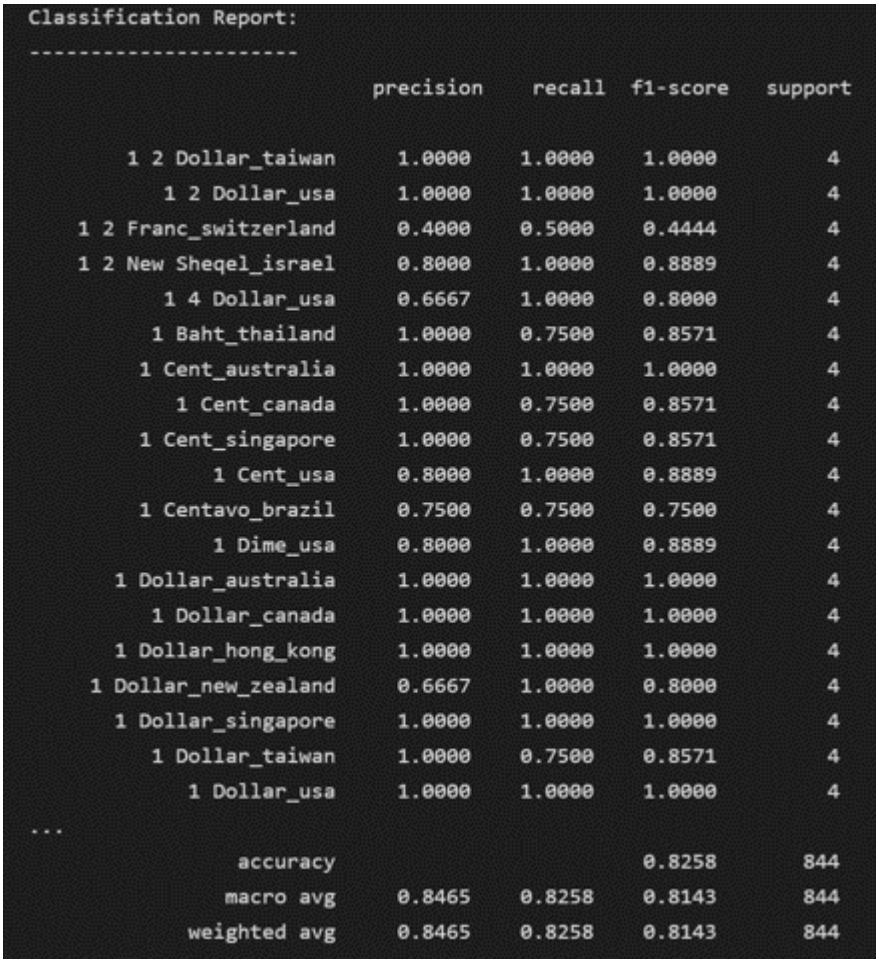
Warstwy połączonej sieci wyglądały następująco:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(500, activation='relu'))
model.add(layers.Dense(211, activation='softmax'))
model.summary()
tf.keras.optimizers.SGD(
    learning_rate=0.01,
    momentum=0.0,
    nesterov=False,
    name='SGD',
)
model.compile(optimizer='Adadelta', loss="sparse_categorical_crossentropy", metrics=['
    accuracy'])#%%
model.fit(X_train, y_train, epochs=8)
```

Ten rodzaj sieci zadziałał najlepiej. Już od pierwszej generacji na zbiorze treningowym ograniczonym osiągał 0.99acc. Na zbiorze testowym 0.92. Po ponownym uczeniu większym zbiorem, udało się osiągnąć pełną dokładność! Przestawiam poniżej krzywe uczenia.

9 Wnioski

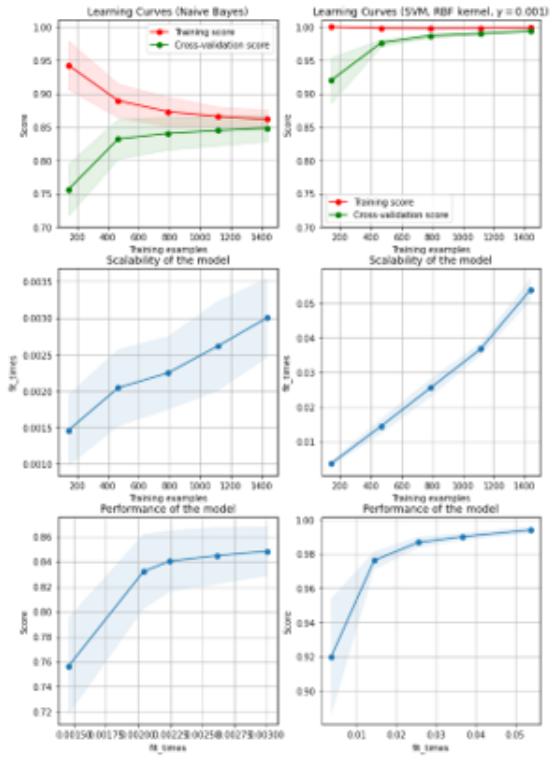
- Najbardziej klasyczny algorytm – MLP – sprawia najwięcej problemów dla obrazów – inaczej dla danych liczbowych lub małej ilości klas



```
Classification Report:
----------------------
                          precision    recall  f1-score   support

       1 2 Dollar_taiwan     1.0000    1.0000    1.0000         4
          1 2 Dollar_usa     1.0000    1.0000    1.0000         4
    1 2 Franc_switzerland    0.4000    0.5000    0.4444         4
    1 2 New Sheqel_israel    0.8000    1.0000    0.8889         4
          1 4 Dollar_usa     0.6667    1.0000    0.8000         4
        1 Baht_thailand      1.0000    0.7500    0.8571         4
       1 Cent_australia      1.0000    1.0000    1.0000         4
          1 Cent_canada      1.0000    0.7500    0.8571         4
       1 Cent_singapore      1.0000    0.7500    0.8571         4
             1 Cent_usa      0.8000    1.0000    0.8889         4
       1 Centavo_brazil      0.7500    0.7500    0.7500         4
             1 Dime_usa      0.8000    1.0000    0.8889         4
      1 Dollar_australia     1.0000    1.0000    1.0000         4
         1 Dollar_canada     1.0000    1.0000    1.0000         4
      1 Dollar_hong_kong     1.0000    1.0000    1.0000         4
    1 Dollar_new_zealand     0.6667    1.0000    0.8000         4
      1 Dollar_singapore     1.0000    1.0000    1.0000         4
         1 Dollar_taiwan     1.0000    0.7500    0.8571         4
            1 Dollar_usa     1.0000    1.0000    1.0000         4
...
                accuracy                         0.8258       844
               macro avg     0.8465    0.8258    0.8143       844
            weighted avg     0.8465    0.8258    0.8143       844
```

Rysunek 12: Raport z klasyfikacji



Rysunek 13: Krzywe uczenia sieci CNN połączonej z siecią MLP.

- Większość metod z publikacji skupiających się na metodach data mining, jeszcze przed zastosowaniem NN nie da lepszego wyniku, niż gotowy model z Kerasa
- Docelowo – w przyszłości mając potrzebę wykorzystania sieci neuronowej, z dużo większym prawdopodobieństwem sięgnę po sieci głębokie – te metody są prostsze w implementacji, mimo bycia bardziej zaawansowanym
- Modele przygotowane z myślą o klasyfikacji tylko dwóch klas, sprawdzają się również dla skomplikowanych datasetów, takich jak mój – 211 klas – i mogą dawać one jeszcze lepsze wyniki…

Dodatkowo pokazuję wyniki różnych sieci, dla problemu klasyfikacji moment. Niestety CNN tutaj zdecydowanie wygrywa.

Sr. No.	Year	Technique Used	Dataset of Coins Used	Modern/Ancient Coins		Accuracy Achieved
				Modern	Ancient	
1	1992 [1]	Neural network, circular array	500 yen (Japan) and 500 won (Korea) coin	√		
2	1993 [2]	Neural network using Genetic Algorithm	500 yen (Japan) and 500 won (Korea) coin	√		
3	1996 [3]	Decision trees	Canadian and Hong Kong coins	√		99.7% - Canadian coins, 98.3% - Hong Kong coins
4	2003 [4]	Edge angle distribution, edge distance distribution	Coins from 30 countries	√		99.24%
5	2004 [5]	Vector Quantization and Histogram Modeling	US coins	√		94%
6	2005 [6]	Eigenspaces and Bayesian fusion	Coins from 30 countries	√		93.23%
7	2005 [7]	Neural network, Gabor filter, Statistical color threshold	Indian coins	√		92.43%
8	2006 [8]	Edge angle distribution, edge distance distribution, edge angle-distance distribution	MUSCLE CIS dataset	√		72%
9	2006 [9, 10]	Neural Network, Pattern Averaging	Turkish 1 Lira and 2 Euro coin	√		96.3%
10	2006 [11]	Multi-scale edge angle histograms, Multi-scale edge distance histograms, Gabor wavelet, Daubechies wavelet	MUSCLE CIS dataset and Merovingen coin dataset	√	√	76%
11	2006 [12]	Fourier approximation of polar image	Thai amulet and Thai baht coins	√		
12	2007 [14]	Tree structured wavelet transform, Ant colony optimization algorithm			√	
13	2007 [13]	Registration approach based on gradient directions	CIS Benchmark dataset	√		
14	2007 [15]	Image abstraction and spiral decomposition	COIN BANK	√	√	
15	2007 [16, 17]	Edge based segmentation, Generalized hough transform and K-nearest neighbor algorithm	MUSCLE CIS dataset	√	√	76%
16	2009 [18]	Gabor wavelet, Euclidean distance and nearest neighbor classifier	MUSCLE dataset	√		74.27%
17	2010 [19]	Fourier approximation of polar image, neural network	Chinese coins	√		83%
18	2010 [20]	Statistical approach	Jordanian coins	√		97%
19	2010 [21]	Rotation invariant template matching	Chinese coins	√		80.6%
20	2011 [22]	Image subtraction technique	Indian coins	√		

Rysunek 14: Pórównanie sieci z publikacji z różnych lat