Rozpoznawanie hieroglifów przy użyciu scikit-image

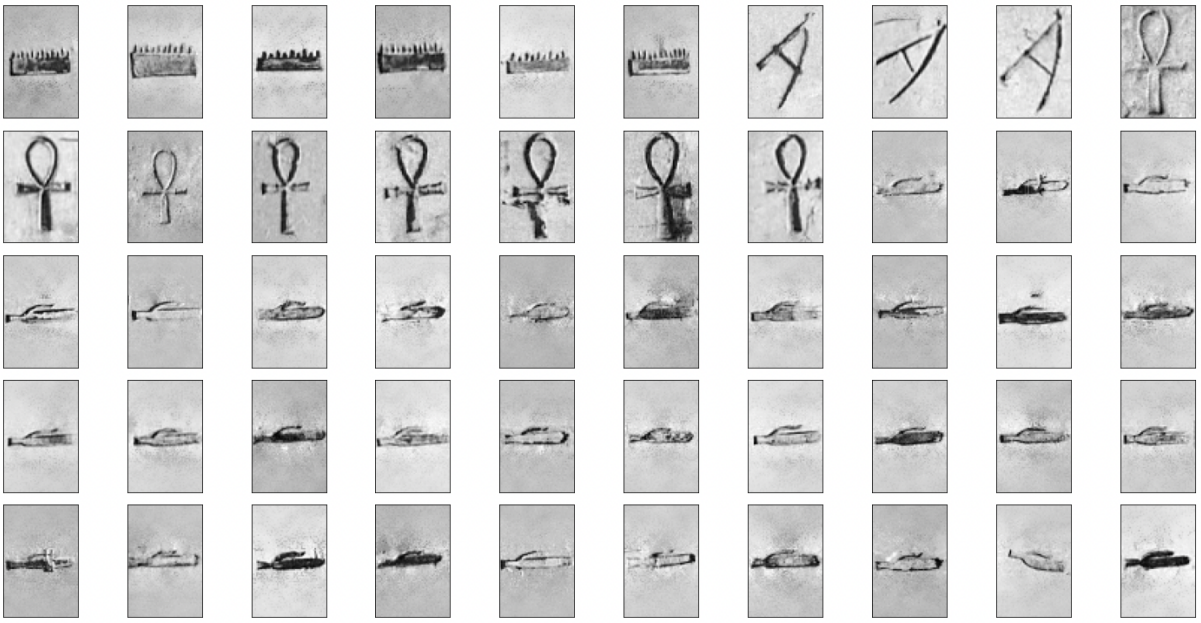
Wydział	Informatyki i Telekomunikacji Politechniki Krakowskiej
Kierunek	Informatyka
Zespół	Aleksandra Bojęś
	Marek Dalida
	Maciej Gicala

1. Cel projektu

Celem projektu było stworzenie systemu umożliwiającego rozpoznawanie hieroglifów. Nauczony model powinien przyjmować na wejście obraz i być w stanie sklasyfikować go jako jeden ze znaków egipskiego pisma obrazkowego.

2. Opis zbioru danych

Dane wykorzystane do nauki modeli są publicznie dostępnym zbiorem zdjęć hieroglifów egipskich, które zachowały się na ścianach Piramidy Unisa.



50 pierwszych obrazów zbioru

W zbiorze jest ponad 3100 obrazów, które należą do czterdziestu klas. Reprezentowane są w odcieniach szarości i mają wymiary 75 na 50 pikseli. Zbiory treningowy i testowy zostały podzielone w proporcji 80/20. W zbiorze występują dysproporcje w liczności obrazów należących do poszczególnych klas - istnieją klasy o tylko kilku obrazach.

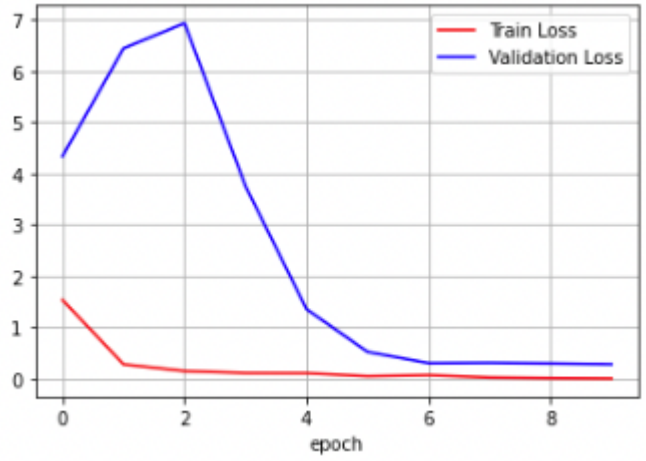
3. Porównanie modeli

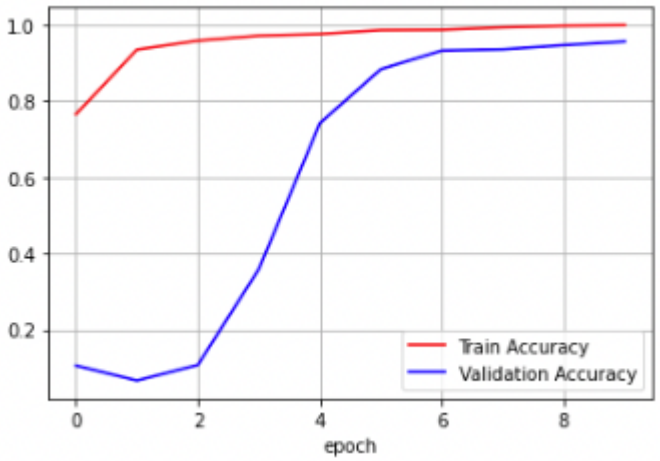
W celu wybrania modelu do testów końcowych, zostały porównane trzy rozwiązania. Wszystkie z nich są neuronowymi sieciami splotowymi. Dwie pierwsze zostały zbudowane przez nas, a trzecia to popularna sieć ResNet50, dostępna między innymi w bibliotece keras.

3.1. Model wyjściowy

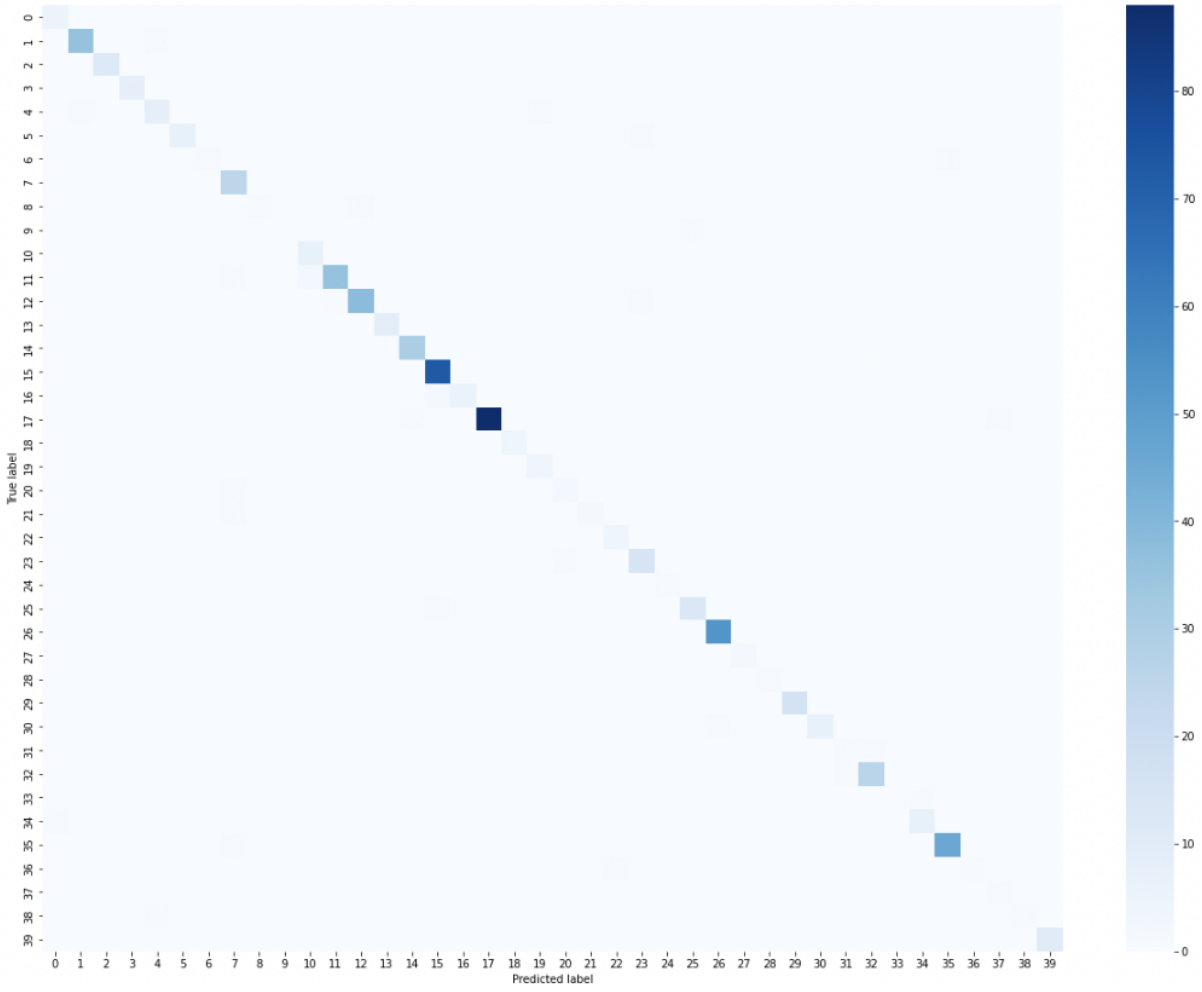
Architektura pierwszej sieci prezentowała się następująco.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 75, 50, 32)	320
activation (Activation)	(None, 75, 50, 32)	0
conv2d_1 (Conv2D)	(None, 73, 48, 32)	9248
activation_1 (Activation)	(None, 73, 48, 32)	0
batch_normalization (BatchNormalization)	(None, 73, 48, 32)	128
max_pooling2d (MaxPooling2D)	(None, 36, 24, 32)	0
dropout (Dropout)	(None, 36, 24, 32)	0
conv2d_2 (Conv2D)	(None, 36, 24, 64)	18496
activation_2 (Activation)	(None, 36, 24, 64)	0
conv2d_3 (Conv2D)	(None, 34, 22, 64)	36928
activation_3 (Activation)	(None, 34, 22, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 34, 22, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 17, 11, 64)	0
dropout_1 (Dropout)	(None, 17, 11, 64)	0
flatten (Flatten)	(None, 11968)	0
dense (Dense)	(None, 392)	4691848
activation_4 (Activation)	(None, 392)	0
dropout_2 (Dropout)	(None, 392)	0
dense_1 (Dense)	(None, 40)	15720
activation_5 (Activation)	(None, 40)	0





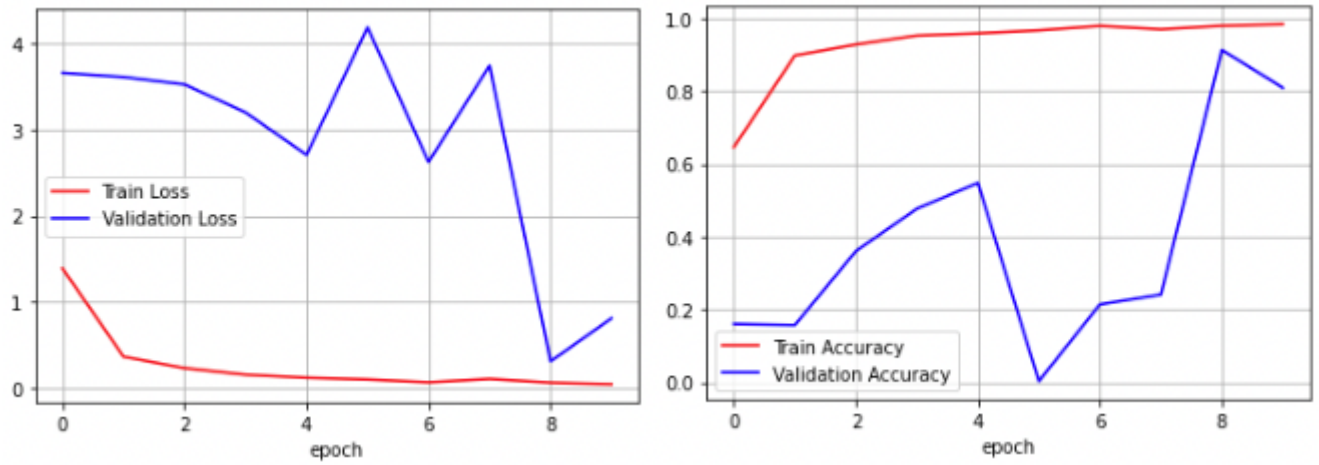
W procesie uczenia już od czwartej epoki widać znaczną poprawę jakości modelu na zbiorze walidacyjnym zarówno jeśli chodzi o wartość funkcji błędu jak i skuteczności. Finalnie na zbiorze testowym model osiągnął 95% skuteczności. Dobre wyniki potwierdza też macierz pomyłek.



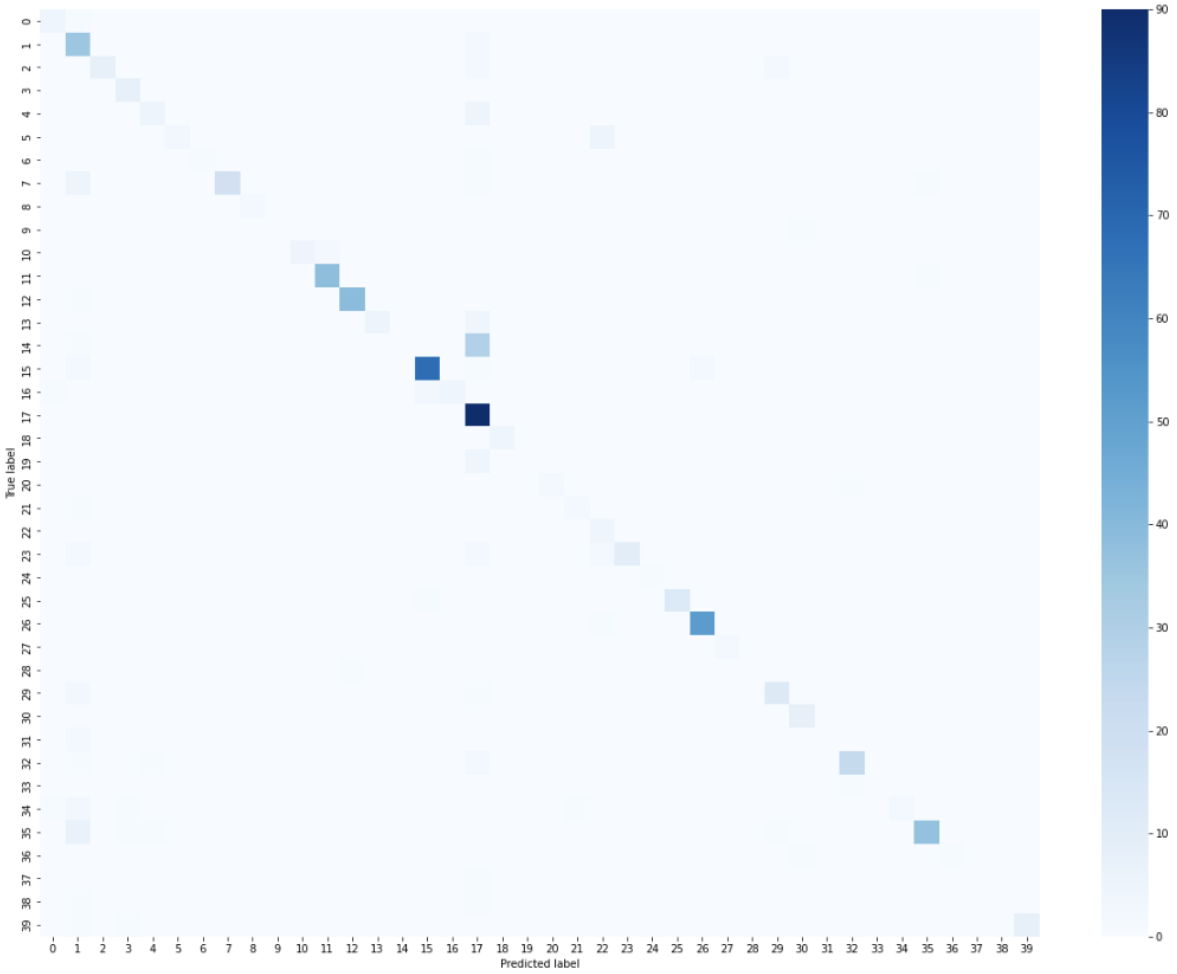
3.2. Model 32x32

Druga sieć miała analogiczną do pierwszej architekturę, z tą różnicą, że na wejście przyjmowała obrazy przeskalowane do wymiarów 32 na 32 piksele. Do przeskalowania zbioru wykorzystana została biblioteka scikit-image.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 32)	320
activation_6 (Activation)	(None, 32, 32, 32)	0
conv2d_5 (Conv2D)	(None, 30, 30, 32)	9248
activation_7 (Activation)	(None, 30, 30, 32)	0
batch_normalization_2 (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_3 (Dropout)	(None, 15, 15, 32)	0
conv2d_6 (Conv2D)	(None, 15, 15, 64)	18496
activation_8 (Activation)	(None, 15, 15, 64)	0
conv2d_7 (Conv2D)	(None, 13, 13, 64)	36928
activation_9 (Activation)	(None, 13, 13, 64)	0
batch_normalization_3 (BatchNormalization)	(None, 13, 13, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_4 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_2 (Dense)	(None, 392)	903560
activation_10 (Activation)	(None, 392)	0
dropout_5 (Dropout)	(None, 392)	0
dense_3 (Dense)	(None, 40)	15720
activation_11 (Activation)	(None, 40)	0



Model na zbiorze walidacyjnym zaczyna osiągać dobre rezultaty od ósmej epoki uczenia. Końcowa skuteczność sieci na zbiorze testowym to około 80%.

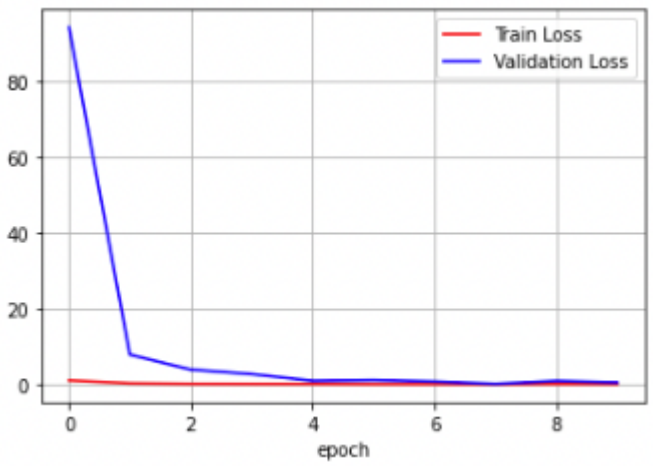


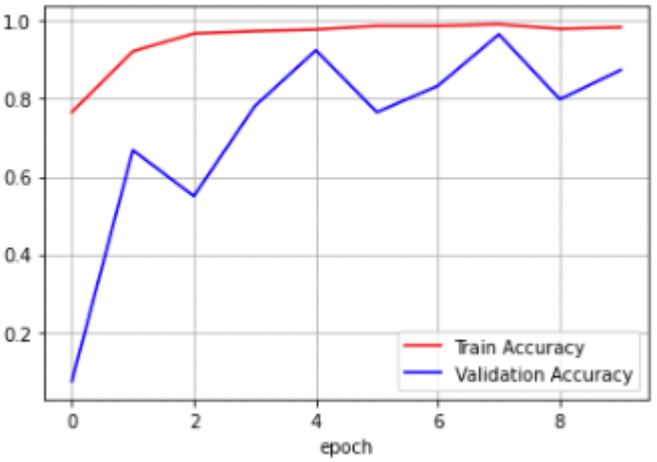
Pogorszenie skuteczności sieci jest też wyraźnie widoczne na macierzy pomyłek. Po zmianie wymiarów danych wejściowych sieć radzi sobie znacznie gorzej z obrazami niektórych klas.

3.3. ResNet50

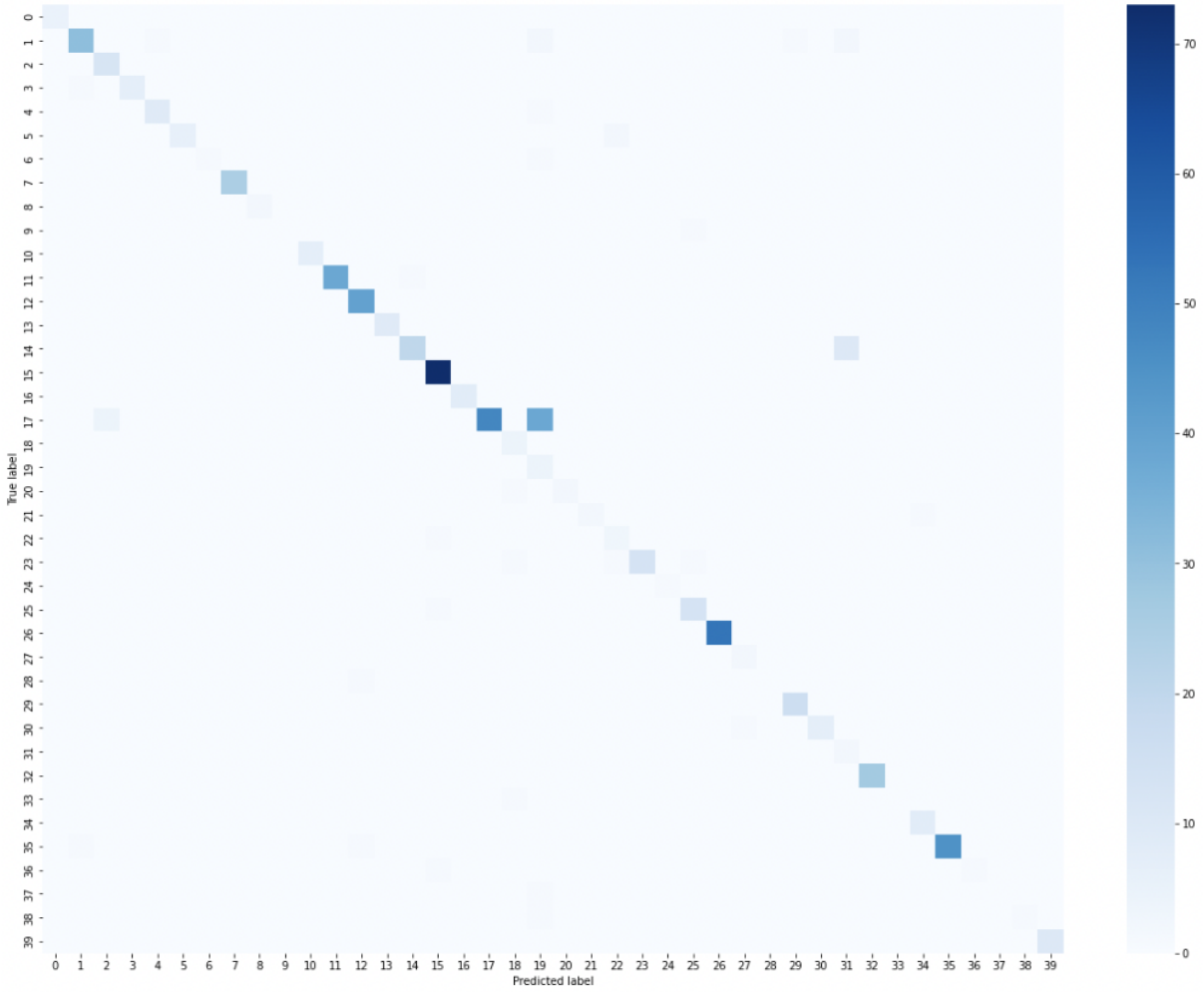
Ostatnim testowanym modelem sieci był ResNet50 dodatkowo rozszerzony. Ze względu na różnicę wymiarów (ResNet przyjmuje obrazy kolorowe - każdy piksel ma trzy wymiary, a wykorzystany zbiór danych jest w odcieniach szarości) przekształcono dane wejściowe tak, aby pasowały do testowanego modelu. Końcowa architektura sieci wyglądała następująco.

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2048)	23587712
dropout_6 (Dropout)	(None, 2048)	0
dense_4 (Dense)	(None, 128)	262272
dropout_7 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 40)	5160





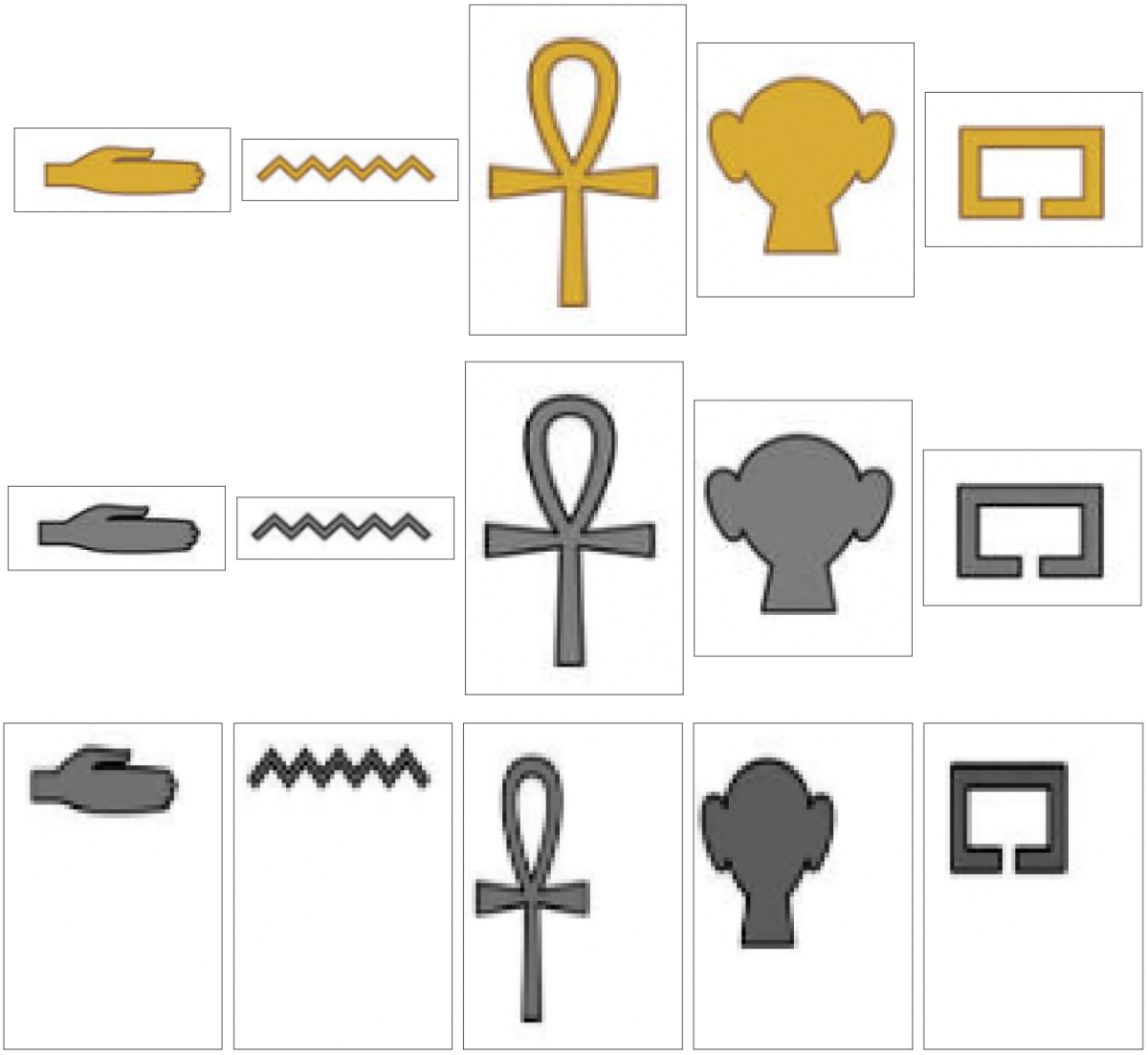
Na zbiorze walidacyjnym sieć osiągała skuteczność około 80% już od trzeciej epoki uczenia. Końcowa skuteczność modelu na zbiorze testowym wyniosła 87%.



Podobnie na podstawie macierzy pomyłek można stwierdzić, że ten model też wypada gorzej niż model wyjściowy. Większość klas klasyfikuje poprawnie, ale ma problem, żeby poprawnie przyporządkować niektóre z obrazów.

4. Testowanie na danych zewnętrznych

Ostatnim etapem projektu było przetestowanie wytrenowanego modelu na danych zewnętrznych - z zupełnie innych źródeł. Badaną siecią był model wyjściowy, ze względu na najlepsze rezultaty w dotychczasowych testach.



Obrazy testowe zostały przekonwertowane do odcieni szarości i przeskalowane do odpowiednich rozmiarów przy użyciu scikit-image.
Niestety na tych danych klasyfikator poradził sobie słabo, poprawnie rozpoznając tylko jeden z badanych obrazów.

```
Expected:  ['D46', 'N35', 'S34', 'D2', 'O1']
Predicted:  ['F31', 'U7', 'S34', 'V30', 'G17']
```