Convolutional neural networks for fashion classification

Sztuczna Inteligencja

Maria Guz, Karol Pątko

16.06.2020

Streszczenie

Niniejszy tekst jest dokumentacją projektu zaliczeniowego z przedmiotu Sztuczna Inteligencja. Zawiera on przykłady wytrenowanych przez studentów modeli sieci splotowych, opisy, wyniki oraz wnioski. Służy przedstawieniu modeli oraz ich porównaniu. Zawiera analizę eksploracyjną i wizualizacje danych, krótkie wprowadzenie teoretyczne na temat machine learningu oraz sieci splotowych i w części właściwej przebieg badań symulacyjnych. Na ostatnich stronach znajduje się podsumowanie projektu. Do dokumentacji dołączony został kod programu (Dodatek A).

Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis danych wejściowych

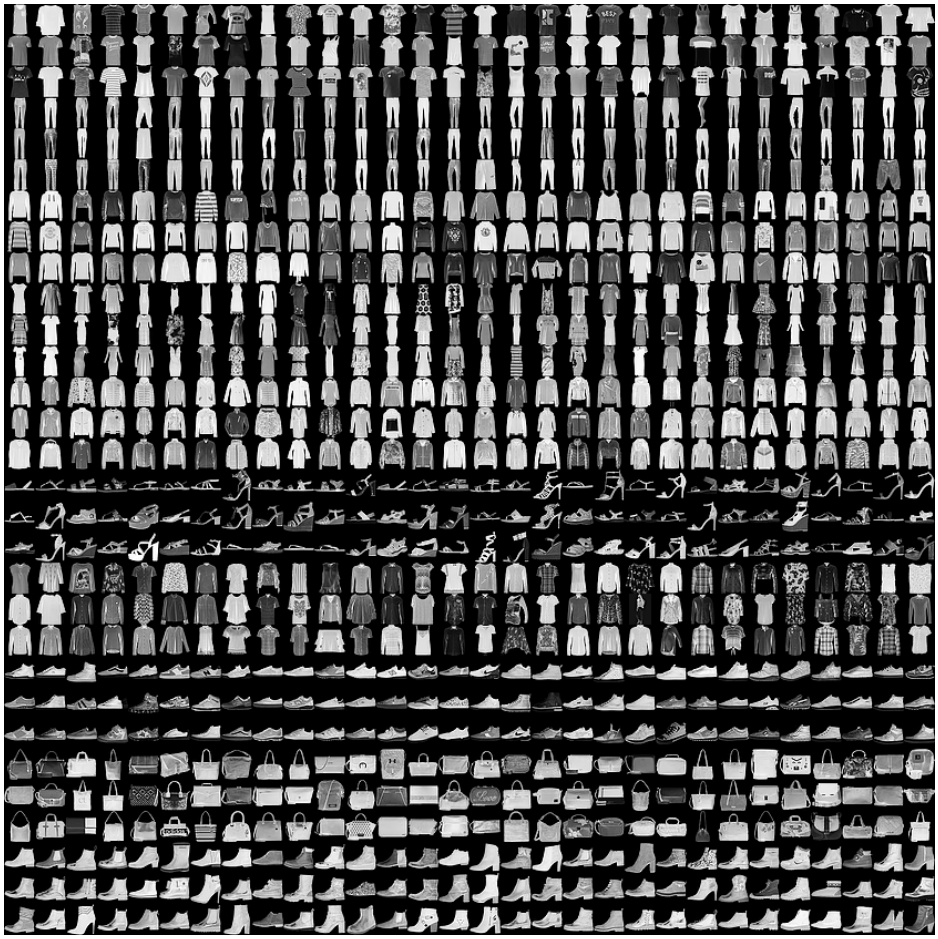
Problemem jest klasyfikacja elementu garderoby ukazanego za pomocą obrazka w skali szarości 28px na 28px znajdującego się w zbiorze danych Fashion-MNIST.

Klasy

- T-shirt/top
- Trouser
- Pullover
- Dress
- Coat
- Sandal
- Shirt
- Sneaker
- Bag
- Ankle boot

1.1.1 Analiza eksploracyjna

Zestaw ten składa się z 70000 elementów - 60000 przykładów trenujących oraz 10000 przykładów testujących. Każda klasa zawiera 6000 elementów trenujących i 1000 elementów testujących. Każdy element zbioru uczącego składa się z wyżej opisanego obrazka oraz jest przypisany do odpowiedniej



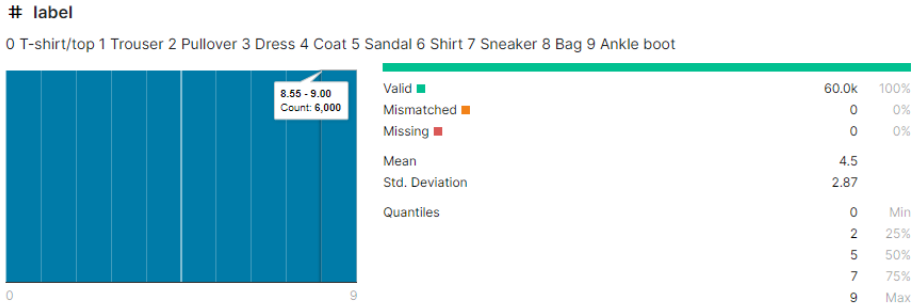
Rysunek 1.1: *Źródło: medium.com*

klasy. Obrazek jest reprezentowany przez 784 kolumny, z której każda przechowuje informację na temat jasności danego piksela (0-255). Stąd możemy zauważyć, że każdy element ma 785 cech. Źródłem obrazków jest Zalando.

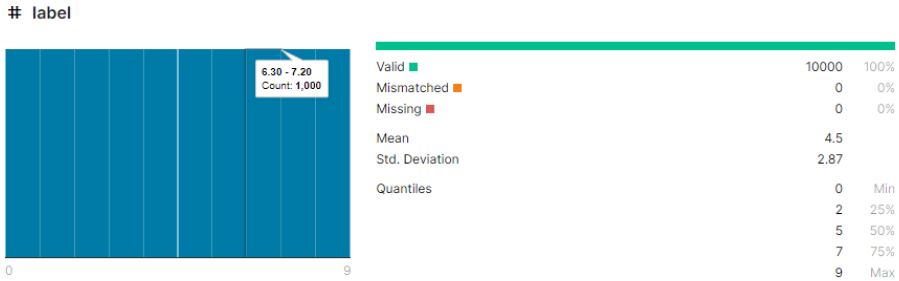
Wymiary zestawu

```
data_train: (60000, 28, 28)
data_test: (10000, 28, 28)
```

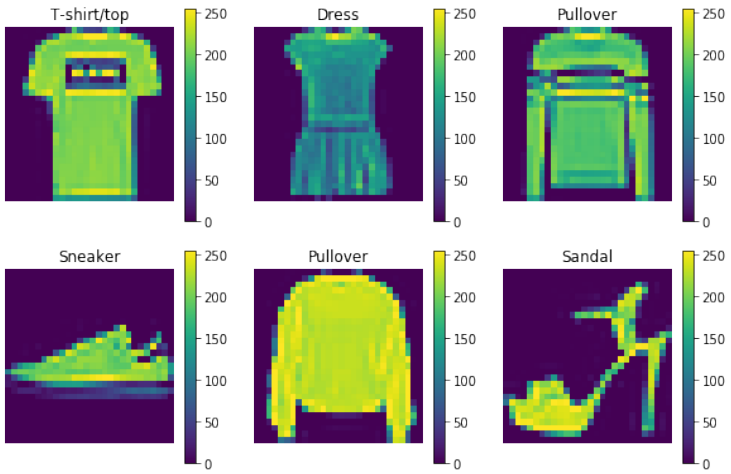
1.1.2 Wizualizacja danych



Rysunek 1.2: *Źródło: kaggle.com*



Rysunek 1.3: *Źródło: kaggle.com*



Rysunek 1.4: *Źródło: własne*

Rozdział 2

Opis metody

2.1 Wprowadzenie teoretyczne

2.1.1 Uczenie maszynowe

Uczenie maszynowe to budowanie modelu matematycznego i wykorzystanie metod matematycznych do przewidywania i samodoskonalenia się. Często takie programy wykorzystuje się do rozpoznawania mowy lub pisma.

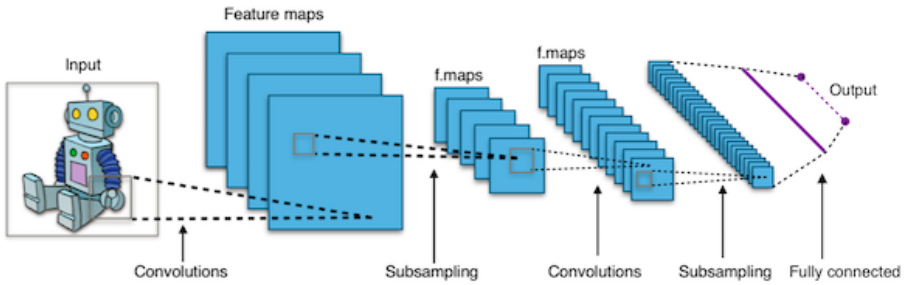
Rodzaje uczenia maszynowego

- **Uczenie nadzorowane** (*Supervised Learning*)
  Program uczy się na podstawie otrzymanych danych oraz odpowiedzi do nich. Ucząc się program generuje pewien wzorzec (model), który wykorzystywany jest później do przewidywania wyników na podstawie innych danych.

- **Uczenie częściowo nadzorowane** (*Semi - Supervised Learning*)
  Program otrzymuje zarówno dane wejściowe zawierające odpowiednie dane wyjściowe, jak i nieoznaczone czyli bez odpowiadającym im wyników.

- **Uczenie nienadzorowane** (*Unsupervised Learning*)
  Program nie posiada „klucza odpowiedzi" i musi sam analizować dane, szukać wzorców i odnajdywać relacje. Wraz ze wzrostem zbiorów danych prezentowane wnioski są coraz bardziej precyzyjne.

- **Uczenie wzmocnione** (*Reinforcement Learning*)
  Program otrzymuje gotowy zestaw dozwolonych działań, reguł i stwierdzeń. Działając w ich ramach dokonuje analizy i obserwuje ich skutki. Wykorzystuje reguły w taki sposób, aby osiągnąć pożądany efekt.



Rysunek 2.1: *Źródło: unite.ai*

2.1.2 Sieć splotowa

W głębokim uczeniu splotowa sieć neuronowa jest klasą głębokich sieci neuronowych, najczęściej stosowanych do analizy obrazów wizualnych. Mają zastosowania w rozpoznawaniu obrazów i filmów, systemach rekomendujących, klasyfikacji obrazów czy przetwarzaniu języka naturalnego. Sieci CNN wykorzystują stosunkowo niewielkie przetwarzanie wstępne w porównaniu z innymi algorytmami klasyfikacji obrazów. Sieć CNN uczy się filtrów, które w tradycyjnych algorytmach były tworzone ręcznie. Splotowa sieć neuronowa składa się z warstwy wejściowej i wyjściowej, a także wielu warstw ukrytych. Ukryte warstwy CNN zazwyczaj składają się z warstw splotowych, warstwy RELU, tj. Funkcji aktywacji, warstw puli, warstw w pełni połączonych i warstw normalizacyjnych.

Warstwy splotowe stosują na wejściu operację splotu, przekazując wynik do następnej warstwy. Splot naśladuje reakcję pojedynczego neuronu na bodźce wzrokowe. Każdy neuron splotowy przetwarza dane tylko dla swojego pola recepcyjnego. Operacja splotu zmniejsza liczbę wolnych parametrów, umożliwiając głębszą sieć z mniejszą liczbą parametrów. Rozwiązuje problem znikania lub eksplozji gradientów w szkoleniu tradycyjnych wielowarstwowych sieci neuronowych z wieloma warstwami przy użyciu propagacji wstecznej. Pomysł konwolucji wziął się po części z nauk informatycznych, ale też po części z biologii. Zasada działania konwolucji jest intuicyjna. Wykorzystuje fakt, że dowolny obiekt pozostaje niezmiennie tym samym obiektem niezależnie od pozycji zajmowanej na obrazie.

Działanie sieci splotowej

1. Pierwszym etapem jest podział obrazu na nakładające się na siebie fragmenty przypominające przesuwające się okno. Następnie należy przesuwać sliding window nad każdym fragmentem i uzyskany wynik zapisywać jako oddzielne dane.

2. Każdy tak powstały fragment jest oddzielnie ładowany do sieci neuro-

nowej. Należy pamiętać o utzrymaniu tych samych wag sieci euronowej dla każdgo elementu. Można jednak zaznaczyć szczególny element, jeśli jest o wskazane.

3. Informacja z każdego fragmentu zostanie zapisana na odwzorowującej oryginał siatce. Tym sposobem otrzymaliśmy zbiór elementów tworzących całość, które ja odwzorowują. Niektóre z tych elementów mogą być zaznaczone jako bardziej interesujące.

4. Zmniejszenie ilości danych może zostać zrealizowane za pomocą algorytmu o nazwie *warstwa zbiorcza*, czyli *max pooling*. Dzięki temu z każdego wyznaczonego fragmentu zostały tylko największe numery.

5. uzyskany w ten sposób zbiór cyfr służy jako dane wejściowe do kolejnej sieci neuronowej. Ostatnia z nich zdecyduje z jakim wzorcem zgadza się obrazek. Jest to *sieć całkowicie podłączona*.

Mamy wiec do czynienia z kolejnymi etapami przetwarzania danych:

- konwolucji,
- warstwy zbiorczej,
- końcowej całkowicie połączonej sieci neuronowej.

Oczywiście ilości konkretych warstw nie ograniczają się do jednej. Można użyć wielu warstw konwolucyjnych, zbiorczych, dyskryminacja niektórych danych. Dodawanie większej liczby warstw konwolucyjnych jest równoznaczne z większą ilością cech, które głęboka sieć neuronowa będzie w stanie rozpoznać. Między innymi to właśnie postaramy się pokazać w niniejszym dokumencie.

Rozdział 3

Badania symulacyjne

3.1 Normalizacja danych

```
num_classes = 10

target_train = keras.utils.to_categorical(target_train, num_classes)
target_test = keras.utils.to_categorical(target_test, num_classes)

data_train = data_train.astype('float32')
data_test = data_test.astype('float32')

#normalize data
data_train /= 255.0
data_test /= 255.0

data_train=data_train.reshape(data_train.shape[0], *(28,28,1))
data_test=data_test.reshape(data_test.shape[0], *(28,28,1))
```

3.2 Dropout

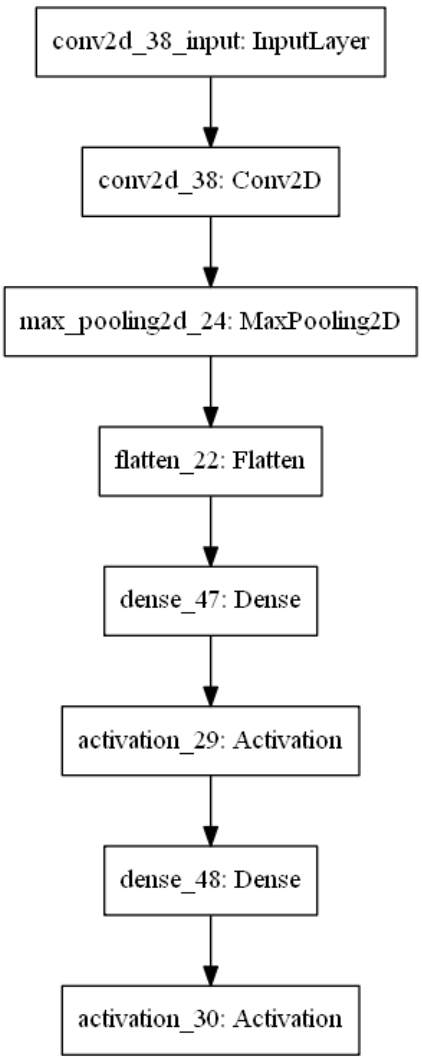
Dropout jest techniką regularyzacji mającą na celu ograniczenie overfittingu w sieciach neuronowych. Jest to skuteczny sposób przeprowadzania uśredniania modelu w sieciach neuronowych. Termin odnosi się do przypadkowego „opuszczenia" lub pominięcia jednostek (zarówno ukrytych, jak i widocznych) podczas procesu uczenia sieci neuronowej.

Jedna warstwa *Conv2D*(rys.2.2)

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_38 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
flatten_22 (Flatten)         (None, 5408)              0
_________________________________________________________________
dense_47 (Dense)             (None, 100)               540900
_________________________________________________________________
activation_29 (Activation)   (None, 100)               0
_________________________________________________________________
dense_48 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_30 (Activation)   (None, 10)                0
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

Jedna warstwa *Conv2D* + dropout (rys. 2.3)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_55 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_45 (Dropout)         (None, 13, 13, 32)        0
_________________________________________________________________
flatten_31 (Flatten)         (None, 5408)              0
_________________________________________________________________
dense_65 (Dense)             (None, 100)               540900
_________________________________________________________________
activation_47 (Activation)   (None, 100)               0
_________________________________________________________________
dense_66 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_48 (Activation)   (None, 10)                0
=================================================================
Total params: 542,230
```



Rysunek 3.1: *Źródło: własne*

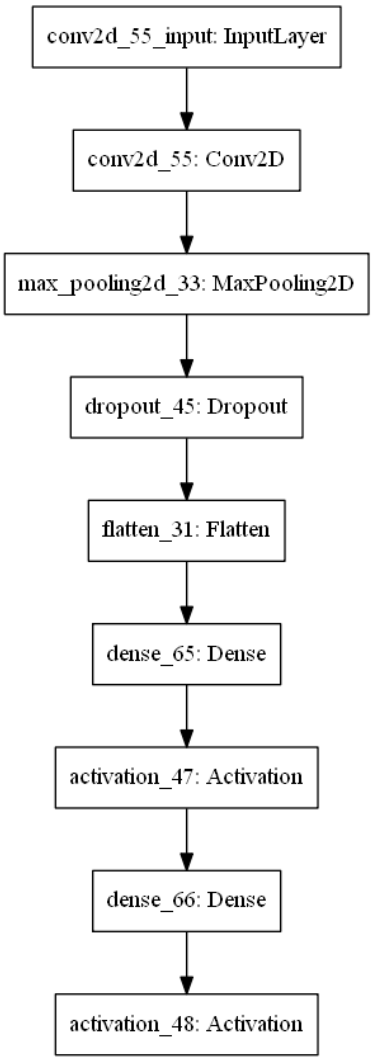
```
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

Więcej warstw *Conv2D* (rys. 2.4)

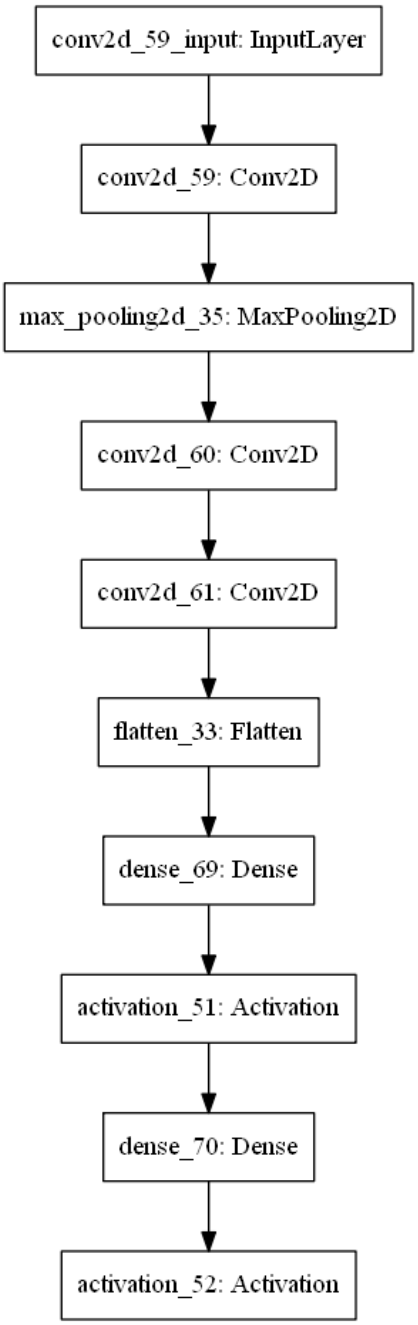
```
 _________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_59 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 11, 11, 32)        9248
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 9, 9, 32)          9248
_________________________________________________________________
flatten_33 (Flatten)         (None, 2592)              0
_________________________________________________________________
dense_69 (Dense)             (None, 100)               259300
_________________________________________________________________
activation_51 (Activation)   (None, 100)               0
_________________________________________________________________
dense_70 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_52 (Activation)   (None, 10)                0
=================================================================
Total params: 279,126
Trainable params: 279,126
Non-trainable params: 0
_________________________________________________________________
```

Więcej warstw *Conv2D* + dropouty(rys. 2.5)

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_62 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_49 (Dropout)         (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 11, 11, 32)        9248
```



Rysunek 3.2: *Źródło: własne*

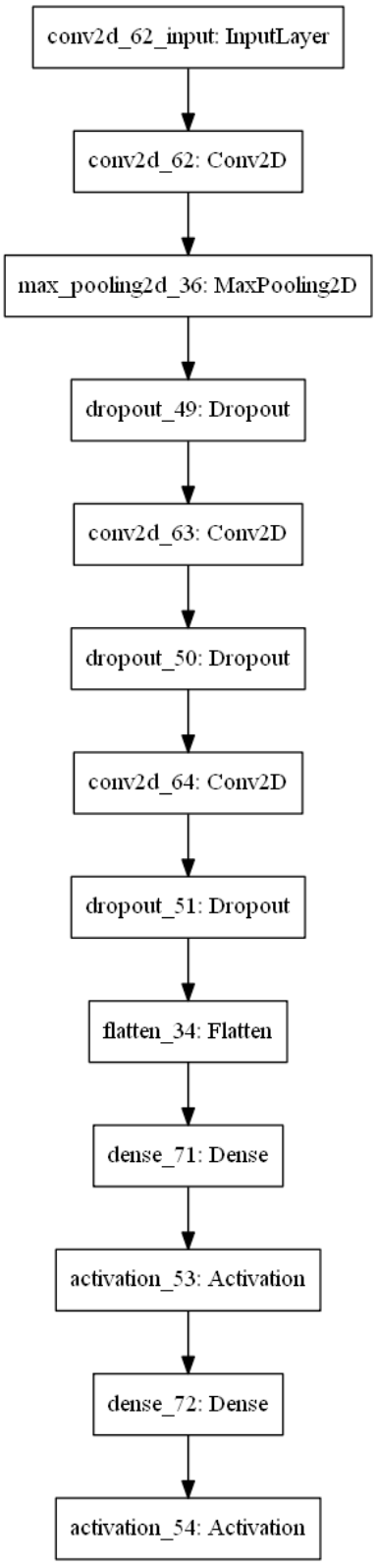


Rysunek 3.3: *Źródło: własne*

```
_________________________________________________________________
dropout_50 (Dropout)         (None, 11, 11, 32)        0
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 9, 9, 32)          9248
_________________________________________________________________
dropout_51 (Dropout)         (None, 9, 9, 32)          0
_________________________________________________________________
flatten_34 (Flatten)         (None, 2592)              0
_________________________________________________________________
dense_71 (Dense)             (None, 100)               259300
_________________________________________________________________
activation_53 (Activation)   (None, 100)               0
_________________________________________________________________
dense_72 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_54 (Activation)   (None, 10)                0
=================================================================
Total params: 279,126
Trainable params: 279,126
Non-trainable params: 0
_________________________________________________________________
```

Więcej warstw *Conv2D* oraz *Dense* + dropouty (rys. 2.6)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_65 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_52 (Dropout)         (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 11, 11, 32)        9248
_________________________________________________________________
dropout_53 (Dropout)         (None, 11, 11, 32)        0
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 9, 9, 32)          9248
_________________________________________________________________
dropout_54 (Dropout)         (None, 9, 9, 32)          0
_________________________________________________________________
flatten_35 (Flatten)         (None, 2592)              0
_________________________________________________________________
```



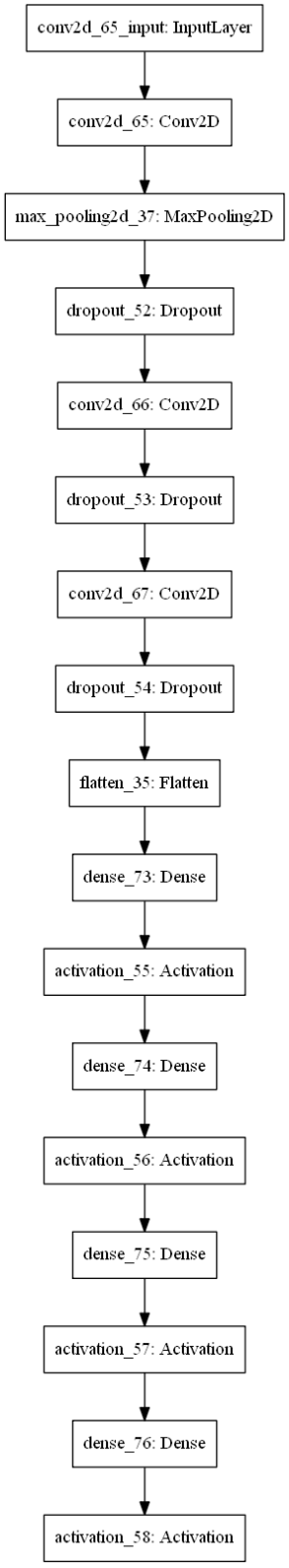
Rysunek 3.4: *Źródło: własne*

```
dense_73 (Dense)             (None, 1000)              2593000
_________________________________________________________________
activation_55 (Activation)   (None, 1000)              0
_________________________________________________________________
dense_74 (Dense)             (None, 500)               500500
_________________________________________________________________
activation_56 (Activation)   (None, 500)               0
_________________________________________________________________
dense_75 (Dense)             (None, 100)               50100
_________________________________________________________________
activation_57 (Activation)   (None, 100)               0
_________________________________________________________________
dense_76 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_58 (Activation)   (None, 10)                0
=================================================================
Total params: 3,163,426
Trainable params: 3,163,426
Non-trainable params: 0
_________________________________________________________________
```

3.2.1 Wyniki

convNN 1

```
    Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 14s 237us/step - loss: 0.9962 - accuracy: 0.6577
- val_loss: 0.7132 - val_accuracy: 0.7167
Epoch 2/10
60000/60000 [==============================] - 14s 239us/step - loss: 0.6400 - accuracy: 0.7665
- val_loss: 0.6082 - val_accuracy: 0.7776
Epoch 3/10
60000/60000 [==============================] - 15s 252us/step - loss: 0.5614 - accuracy: 0.7930
- val_loss: 0.5772 - val_accuracy: 0.7825
Epoch 4/10
60000/60000 [==============================] - 16s 260us/step - loss: 0.5211 - accuracy: 0.8097
- val_loss: 0.5090 - val_accuracy: 0.8142
Epoch 5/10
60000/60000 [==============================] - 15s 258us/step - loss: 0.4944 - accuracy: 0.8203
- val_loss: 0.5048 - val_accuracy: 0.8103
Epoch 6/10
60000/60000 [==============================] - 16s 260us/step - loss: 0.4713 - accuracy: 0.8307
- val_loss: 0.4745 - val_accuracy: 0.8276
Epoch 7/10
60000/60000 [==============================] - 16s 267us/step - loss: 0.4522 - accuracy: 0.8378
- val_loss: 0.4799 - val_accuracy: 0.8189
Epoch 8/10
60000/60000 [==============================] - 16s 263us/step - loss: 0.4408 - accuracy: 0.8419
- val_loss: 0.4396 - val_accuracy: 0.8440
Epoch 9/10
60000/60000 [==============================] - 16s 267us/step - loss: 0.4221 - accuracy: 0.8486
```



Rysunek 3.5: *Źródło: własne*

```
- val_loss: 0.4503 - val_accuracy: 0.8351
Epoch 10/10
60000/60000 [==============================] - 16s 262us/step - loss: 0.4144 - accuracy: 0.8510
- val_loss: 0.4346 - val_accuracy: 0.8426
```

convNN 2

```
    Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 17s 281us/step - loss: 1.0228 - accuracy: 0.6586
- val_loss: 0.8121 - val_accuracy: 0.7144
Epoch 2/10
60000/60000 [==============================] - 17s 288us/step - loss: 0.6404 - accuracy: 0.7633
- val_loss: 0.6127 - val_accuracy: 0.7605
Epoch 3/10
60000/60000 [==============================] - 18s 294us/step - loss: 0.5744 - accuracy: 0.7874
- val_loss: 0.5370 - val_accuracy: 0.7975
Epoch 4/10
60000/60000 [==============================] - 18s 300us/step - loss: 0.5363 - accuracy: 0.8027
- val_loss: 0.5066 - val_accuracy: 0.8166
Epoch 5/10
60000/60000 [==============================] - 19s 309us/step - loss: 0.5089 - accuracy: 0.8143
- val_loss: 0.5004 - val_accuracy: 0.8152
Epoch 6/10
60000/60000 [==============================] - 18s 304us/step - loss: 0.4792 - accuracy: 0.8273
- val_loss: 0.4846 - val_accuracy: 0.8183
Epoch 7/10
60000/60000 [==============================] - 19s 309us/step - loss: 0.4631 - accuracy: 0.8335
- val_loss: 0.4675 - val_accuracy: 0.8270
Epoch 8/10
60000/60000 [==============================] - 18s 305us/step - loss: 0.4494 - accuracy: 0.8388
- val_loss: 0.4700 - val_accuracy: 0.8227
Epoch 9/10
60000/60000 [==============================] - 19s 312us/step - loss: 0.4315 - accuracy: 0.8450
- val_loss: 0.4447 - val_accuracy: 0.8391
Epoch 10/10
60000/60000 [==============================] - 20s 335us/step - loss: 0.4167 - accuracy: 0.8524
- val_loss: 0.4294 - val_accuracy: 0.8444
```

convNN 3

```
    Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 25s 417us/step - loss: 1.4088 - accuracy: 0.4902
- val_loss: 0.9269 - val_accuracy: 0.6583
Epoch 2/10
60000/60000 [==============================] - 26s 431us/step - loss: 0.7264 - accuracy: 0.7255
- val_loss: 0.6973 - val_accuracy: 0.7248
Epoch 3/10
60000/60000 [==============================] - 27s 443us/step - loss: 0.6038 - accuracy: 0.7693
- val_loss: 0.6014 - val_accuracy: 0.7670
Epoch 4/10
60000/60000 [==============================] - 26s 436us/step - loss: 0.5405 - accuracy: 0.7955
- val_loss: 0.5716 - val_accuracy: 0.7921
Epoch 5/10
60000/60000 [==============================] - 26s 438us/step - loss: 0.5025 - accuracy: 0.8134
- val_loss: 0.5049 - val_accuracy: 0.8146
Epoch 6/10
60000/60000 [==============================] - 26s 439us/step - loss: 0.4683 - accuracy: 0.8291
- val_loss: 0.5295 - val_accuracy: 0.8086
```

```
Epoch 7/10
60000/60000 [==============================] - 26s 441us/step - loss: 0.4414 - accuracy: 0.8400
- val_loss: 0.4569 - val_accuracy: 0.8347
Epoch 8/10
60000/60000 [==============================] - 27s 443us/step - loss: 0.4253 - accuracy: 0.8461
- val_loss: 0.4414 - val_accuracy: 0.8398
Epoch 9/10
60000/60000 [==============================] - 27s 446us/step - loss: 0.4102 - accuracy: 0.8527
- val_loss: 0.4536 - val_accuracy: 0.8312
Epoch 10/10
60000/60000 [==============================] - 28s 459us/step - loss: 0.3953 - accuracy: 0.8565
- val_loss: 0.4385 - val_accuracy: 0.8393
```

convNN 3a

```
    Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 31s 517us/step - loss: 1.3064 - accuracy: 0.5257
- val_loss: 0.8602 - val_accuracy: 0.6625
Epoch 2/10
60000/60000 [==============================] - 32s 527us/step - loss: 0.7668 - accuracy: 0.7088
- val_loss: 0.6540 - val_accuracy: 0.7405
Epoch 3/10
60000/60000 [==============================] - 32s 537us/step - loss: 0.6553 - accuracy: 0.7487
- val_loss: 0.6614 - val_accuracy: 0.7499
Epoch 4/10
60000/60000 [==============================] - 32s 537us/step - loss: 0.5941 - accuracy: 0.7745
- val_loss: 0.6004 - val_accuracy: 0.7712
Epoch 5/10
60000/60000 [==============================] - 32s 541us/step - loss: 0.5519 - accuracy: 0.7930
- val_loss: 0.5130 - val_accuracy: 0.8116
Epoch 6/10
60000/60000 [==============================] - 33s 549us/step - loss: 0.5214 - accuracy: 0.8057
- val_loss: 0.4985 - val_accuracy: 0.8212
Epoch 7/10
60000/60000 [==============================] - 33s 545us/step - loss: 0.4936 - accuracy: 0.8172
- val_loss: 0.4717 - val_accuracy: 0.8304
Epoch 8/10
60000/60000 [==============================] - 33s 546us/step - loss: 0.4735 - accuracy: 0.8254
- val_loss: 0.4541 - val_accuracy: 0.8397
Epoch 9/10
60000/60000 [==============================] - 33s 544us/step - loss: 0.4525 - accuracy: 0.8340
- val_loss: 0.4499 - val_accuracy: 0.8357
Epoch 10/10
60000/60000 [==============================] - 33s 546us/step - loss: 0.4354 - accuracy: 0.8421
- val_loss: 0.4209 - val_accuracy: 0.8511
```

convNN 4

```
    Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 40s 660us/step - loss: 1.4434 - accuracy: 0.4768
- val_loss: 0.8869 - val_accuracy: 0.6521
Epoch 2/10
60000/60000 [==============================] - 40s 667us/step - loss: 0.8000 - accuracy: 0.6929
- val_loss: 0.6942 - val_accuracy: 0.7375
Epoch 3/10
60000/60000 [==============================] - 41s 687us/step - loss: 0.6657 - accuracy: 0.7449
- val_loss: 0.5950 - val_accuracy: 0.7764
Epoch 4/10
```

```
60000/60000 [==============================] - 43s 722us/step - loss: 0.5973 - accuracy: 0.7706
- val_loss: 0.5890 - val_accuracy: 0.7774
Epoch 5/10
60000/60000 [==============================] - 42s 695us/step - loss: 0.5546 - accuracy: 0.7874
- val_loss: 0.6170 - val_accuracy: 0.7723
Epoch 6/10
60000/60000 [==============================] - 40s 663us/step - loss: 0.5214 - accuracy: 0.8031
- val_loss: 0.4839 - val_accuracy: 0.8196
Epoch 7/10
60000/60000 [==============================] - 40s 661us/step - loss: 0.4932 - accuracy: 0.8123
- val_loss: 0.4579 - val_accuracy: 0.8326
Epoch 8/10
60000/60000 [==============================] - 41s 684us/step - loss: 0.4629 - accuracy: 0.8280
- val_loss: 0.4608 - val_accuracy: 0.8257
Epoch 9/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.4455 - accuracy: 0.8332
- val_loss: 0.4411 - val_accuracy: 0.8354
Epoch 10/10
60000/60000 [==============================] - 45s 754us/step - loss: 0.4244 - accuracy: 0.8414
- val_loss: 0.4089 - val_accuracy: 0.8522
```

3.3 Warstwy splotowe

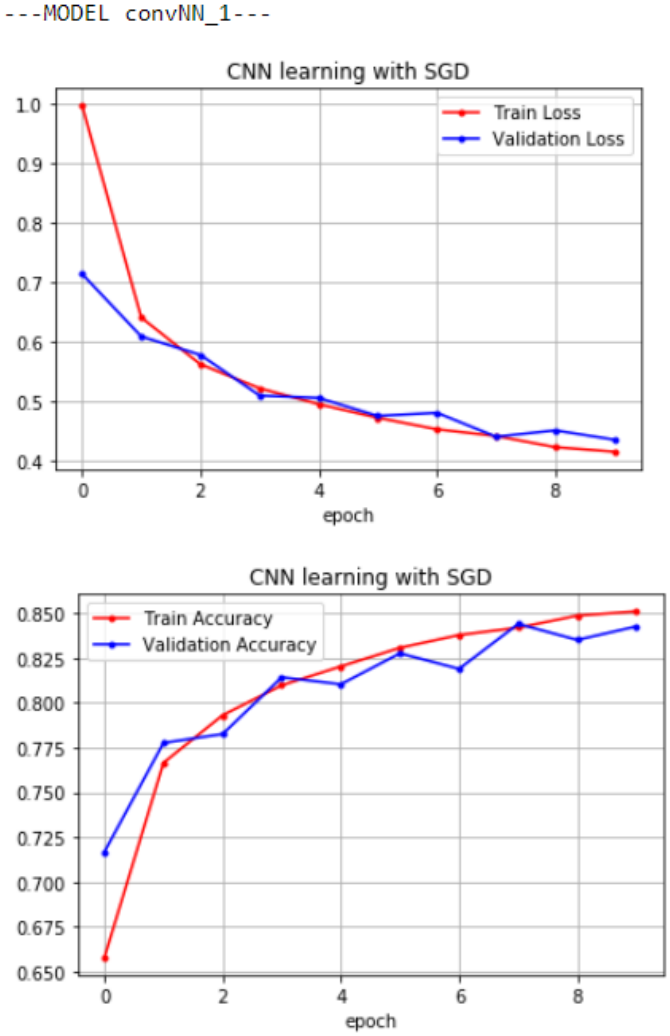
Conv2D layer

Keras Conv2D jest warstwą konwolucji 2D, która tworzy jądro splotu z wiatrem z wejściowymi warstwami, które pomagają wytworzyć tensor wyników.W przetwarzaniu obrazu jądro to matryca lub maski splotowe, które mogą być używane do rozmycia, wyostrzania, wytłaczania, wykrywania krawędzi i nie tylko poprzez wykonanie splotu między jądrem a obrazem.

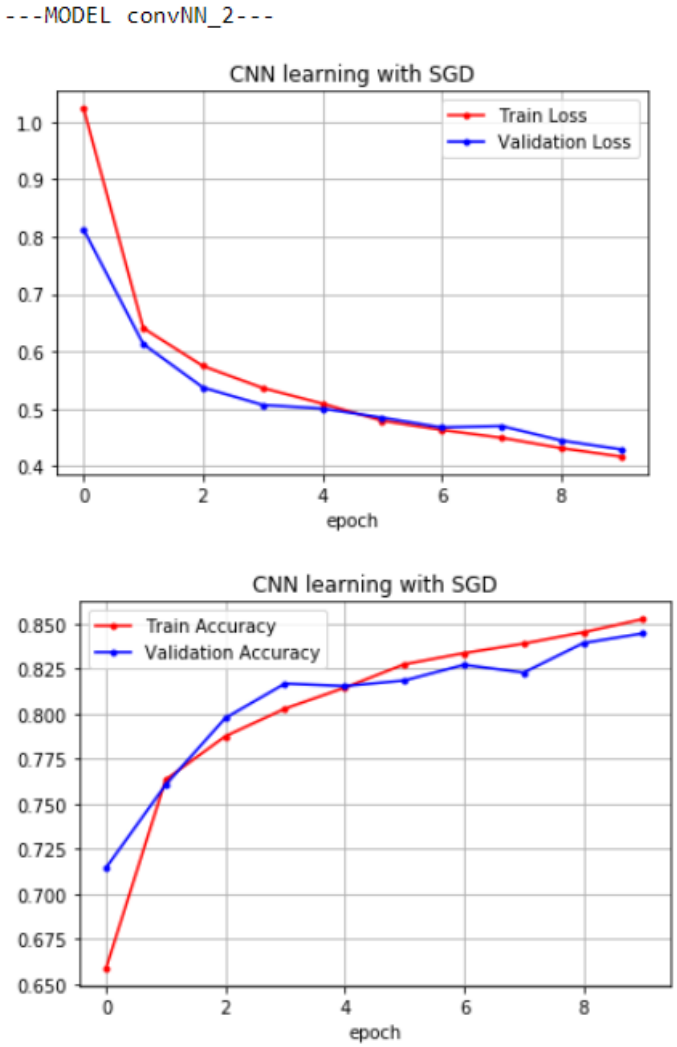
SeparableConv2D layer (rys. 2.12)

SeparableConv2D jest odmianą tradycyjnego splotu, który zaproponowano, aby obliczyć go szybciej. Dokonuje głębokiego splotu przestrzennego, a następnie splotu punktowego, który miesza razem powstałe kanały wyjściowe.

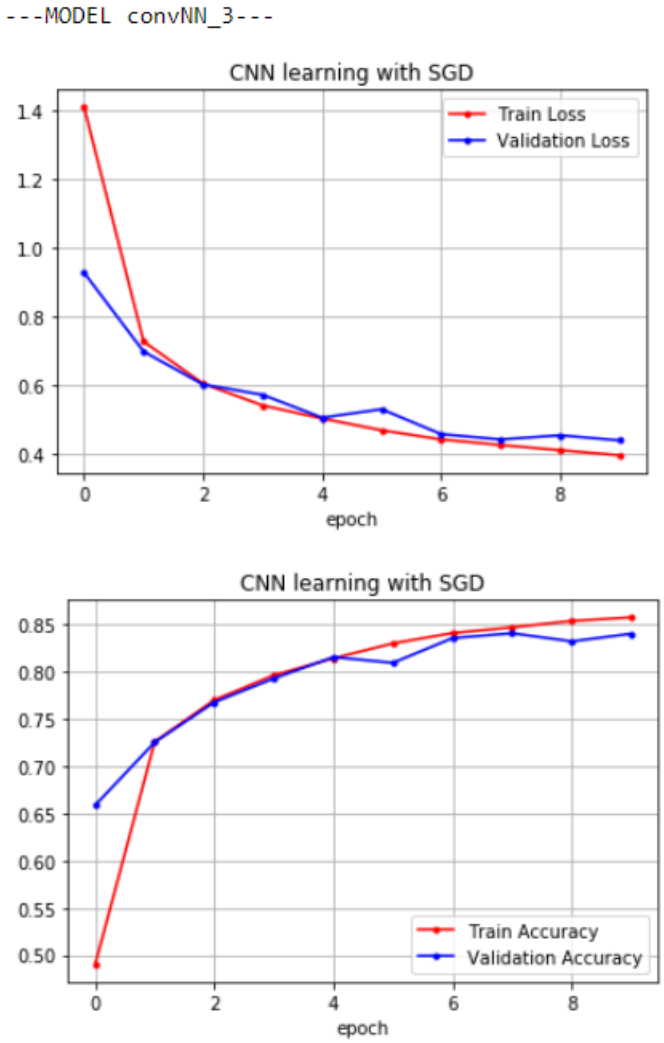
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv2d_4 (Separabl (None, 26, 26, 32)        73
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
flatten_42 (Flatten)         (None, 5408)              0
_________________________________________________________________
dense_89 (Dense)             (None, 100)               540900
```



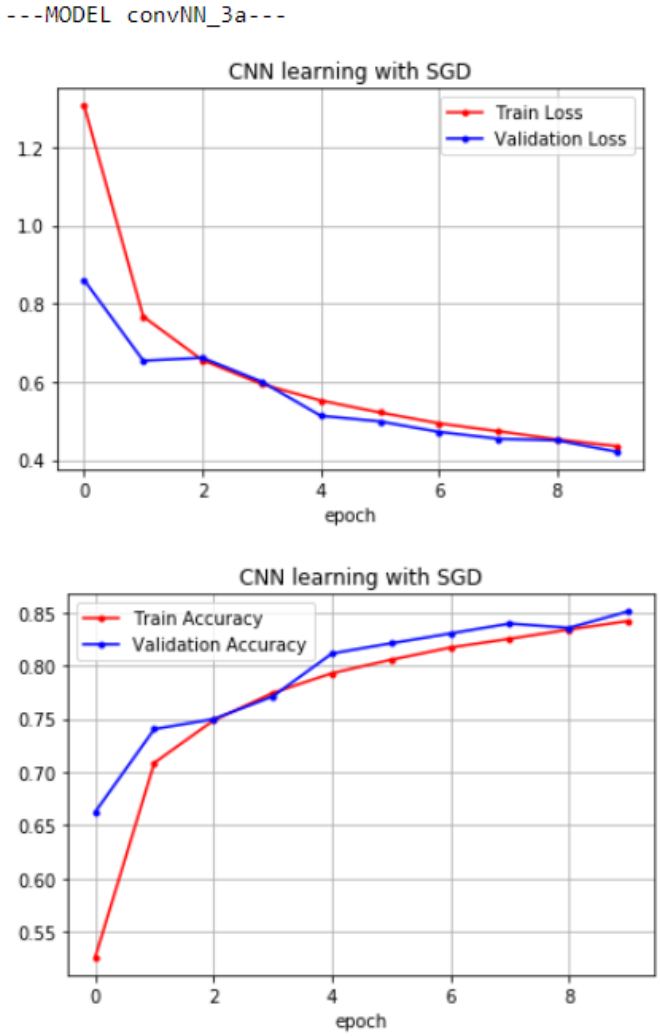
Rysunek 3.6: *Źródło: własne*



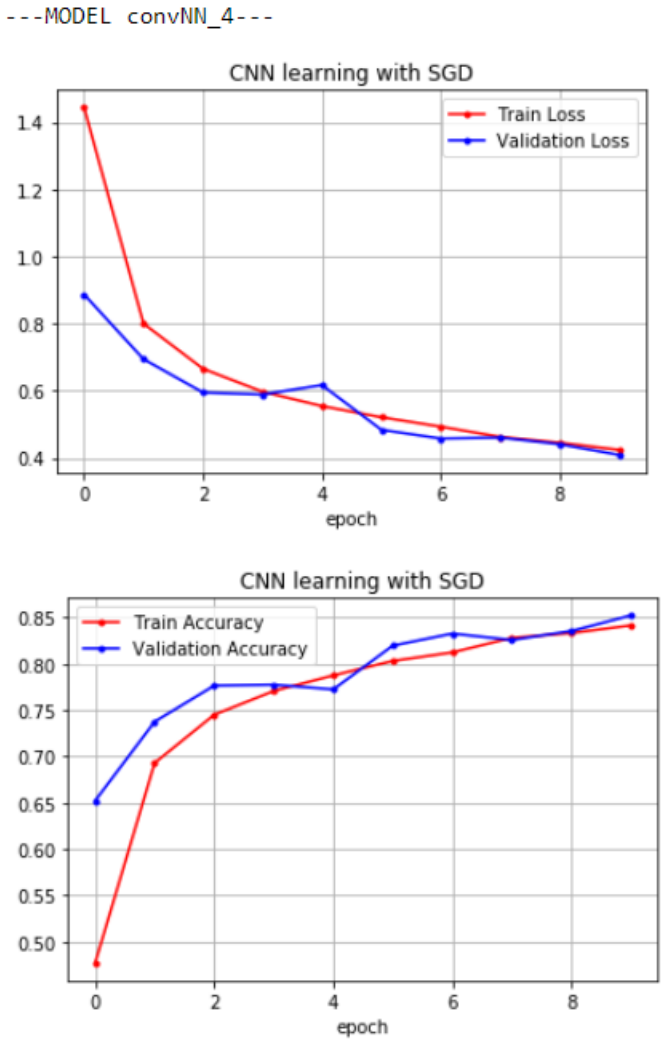
Rysunek 3.7: *Źródło: własne*



Rysunek 3.8: *Źródło: własne*



Rysunek 3.9: *Źródło: własne*



Rysunek 3.10: *Źródło: własne*

```
_________________________________________________________________
activation_71 (Activation)   (None, 100)               0
_________________________________________________________________
dense_90 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_72 (Activation)   (None, 10)                0
=================================================================
Total params: 541,983
Trainable params: 541,983
Non-trainable params: 0
_________________________________________________________________
```

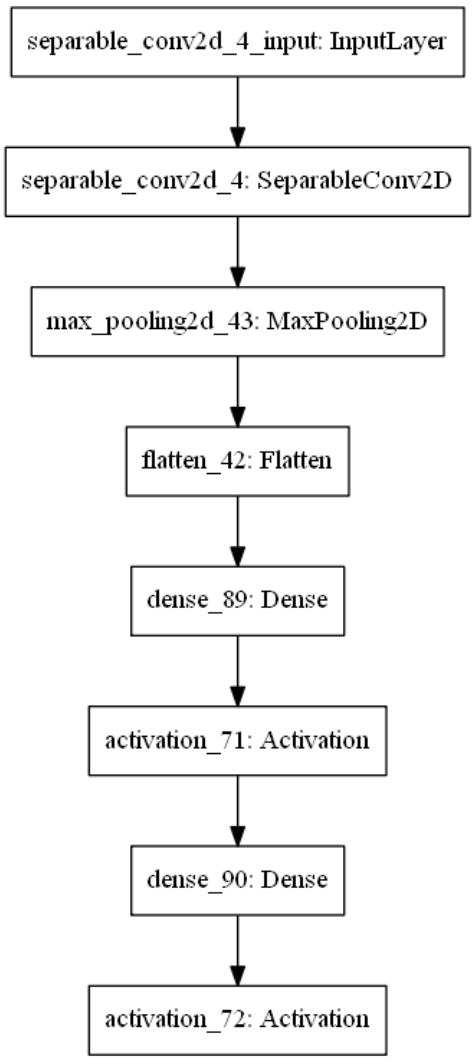
DepthwiseConv2D layer (rys. 2.13)

Depthwise separable convolutions polegają na wykonaniu tylko pierwszego kroku w głębokiej splotie przestrzennej (która działa na każdy kanał wejściowy osobno). Argument *deep multiplier* kontroluje liczbę kanałów wyjściowych generowanych na kanał wejściowy w kroku głębokości.

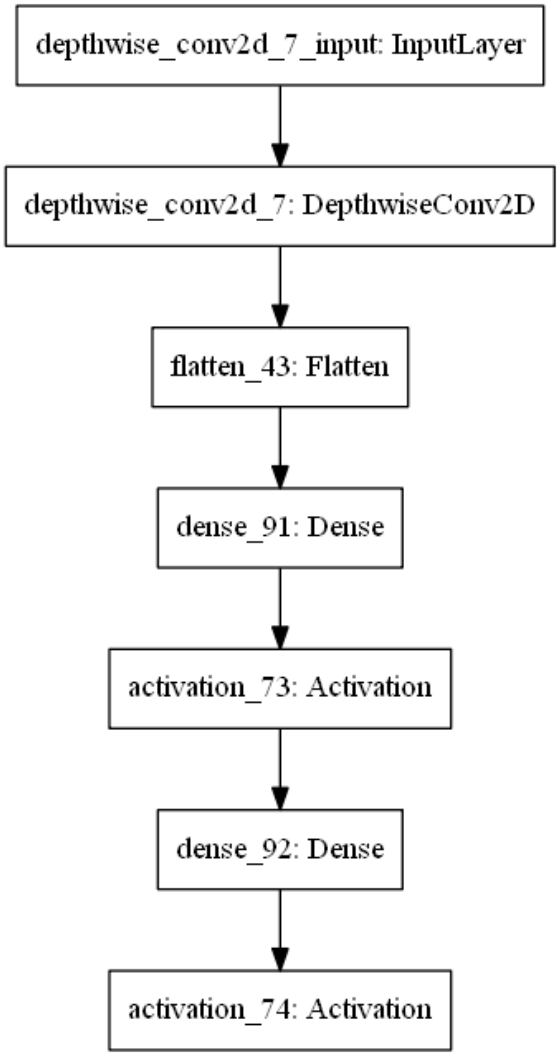
```
 _________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
depthwise_conv2d_7 (Depthwis (None, 1, 1, 1)           785
_________________________________________________________________
flatten_43 (Flatten)         (None, 1)                 0
_________________________________________________________________
dense_91 (Dense)             (None, 100)               200
_________________________________________________________________
activation_73 (Activation)   (None, 100)               0
_________________________________________________________________
dense_92 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_74 (Activation)   (None, 10)                0
=================================================================
Total params: 1,995
Trainable params: 1,995
Non-trainable params: 0
_________________________________________________________________
```

Conv2DTranspose layer (rys. 2.14)

Potrzeba transponowania splotów zasadniczo wynika z chęci zastosowania transformacji zmierzającej w przeciwnym kierunku niż normalny splot, tj. Z



Rysunek 3.11: *Źródło: własne*



Rysunek 3.12: *Źródło: własne*

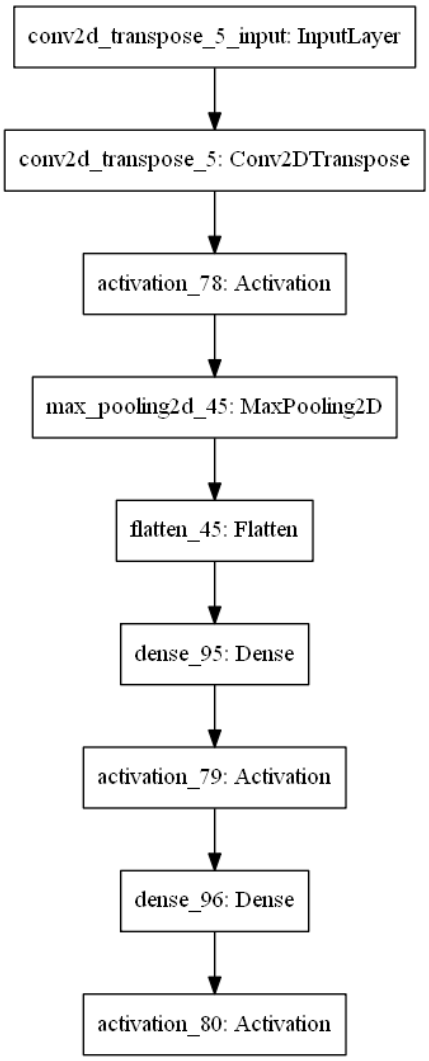
czegoś, co ma kształt wyjścia jakiegoś splotu, do czegoś, co ma kształt jego wejścia, przy zachowaniu, że wzorzec łączności jest zgodny z wymienionym splotem.

```
 _________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_transpose_5 (Conv2DTr (None, 30, 30, 32)        320
_________________________________________________________________
activation_78 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
flatten_45 (Flatten)         (None, 7200)              0
_________________________________________________________________
dense_95 (Dense)             (None, 100)               720100
_________________________________________________________________
activation_79 (Activation)   (None, 100)               0
_________________________________________________________________
dense_96 (Dense)             (None, 10)                1010
_________________________________________________________________
activation_80 (Activation)   (None, 10)                0
=================================================================
Total params: 721,430
Trainable params: 721,430
Non-trainable params: 0
_________________________________________________________________
```

3.3.1 Wyniki

SeparableConv2D

```
    Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 16s 275us/step - loss: 1.2064 - accuracy: 0.5850
- val_loss: 0.7699 - val_accuracy: 0.7129
Epoch 2/10
60000/60000 [==============================] - 18s 292us/step - loss: 0.6479 - accuracy: 0.7629
- val_loss: 0.6117 - val_accuracy: 0.7682
Epoch 3/10
60000/60000 [==============================] - 18s 298us/step - loss: 0.5508 - accuracy: 0.7978
- val_loss: 0.5183 - val_accuracy: 0.8114
Epoch 4/10
60000/60000 [==============================] - 18s 299us/step - loss: 0.4941 - accuracy: 0.8198
- val_loss: 0.4832 - val_accuracy: 0.8262
Epoch 5/10
60000/60000 [==============================] - 18s 300us/step - loss: 0.4560 - accuracy: 0.8356
- val_loss: 0.4560 - val_accuracy: 0.8348
```



Rysunek 3.13: *Źródło: własne*

```
Epoch 6/10
60000/60000 [==============================] - 18s 300us/step - loss: 0.4308 - accuracy: 0.8446
- val_loss: 0.4482 - val_accuracy: 0.8370
Epoch 7/10
60000/60000 [==============================] - 18s 303us/step - loss: 0.4091 - accuracy: 0.8522
- val_loss: 0.4192 - val_accuracy: 0.8522
Epoch 8/10
60000/60000 [==============================] - 18s 304us/step - loss: 0.3950 - accuracy: 0.8577
- val_loss: 0.4026 - val_accuracy: 0.8560
Epoch 9/10
60000/60000 [==============================] - 18s 306us/step - loss: 0.3834 - accuracy: 0.8621
- val_loss: 0.3957 - val_accuracy: 0.8560
Epoch 10/10
60000/60000 [==============================] - 18s 307us/step - loss: 0.3706 - accuracy: 0.8663
- val_loss: 0.4021 - val_accuracy: 0.8526
```

DepthwiseConv2D

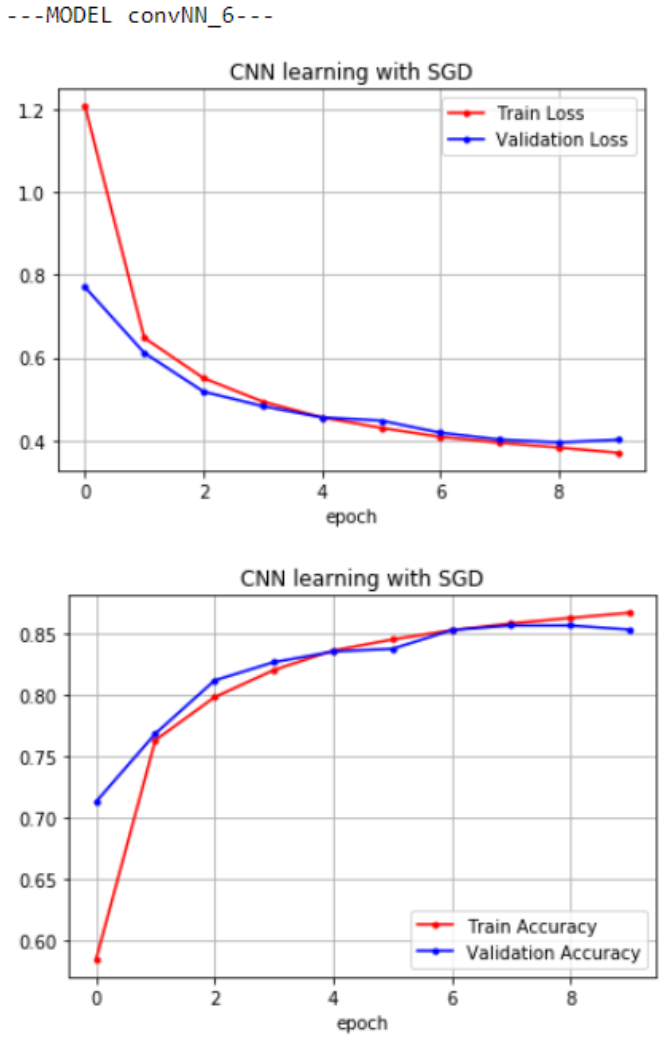
```
 Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 6s 93us/step - loss: 2.1119 - accuracy: 0.1795
- val_loss: 1.9690 - val_accuracy: 0.2079
Epoch 2/10
60000/60000 [==============================] - 5s 86us/step - loss: 1.8988 - accuracy: 0.2238
- val_loss: 1.8404 - val_accuracy: 0.2581
Epoch 3/10
60000/60000 [==============================] - 5s 87us/step - loss: 1.7890 - accuracy: 0.2506
- val_loss: 1.7470 - val_accuracy: 0.2691
Epoch 4/10
60000/60000 [==============================] - 5s 86us/step - loss: 1.7047 - accuracy: 0.2745
- val_loss: 1.6729 - val_accuracy: 0.2968
Epoch 5/10
60000/60000 [==============================] - 5s 87us/step - loss: 1.6347 - accuracy: 0.2936
- val_loss: 1.6143 - val_accuracy: 0.3262
Epoch 6/10
60000/60000 [==============================] - 5s 87us/step - loss: 1.5747 - accuracy: 0.3123
- val_loss: 1.5553 - val_accuracy: 0.3173
Epoch 7/10
60000/60000 [==============================] - 5s 87us/step - loss: 1.5250 - accuracy: 0.3280
- val_loss: 1.5184 - val_accuracy: 0.3504
Epoch 8/10
60000/60000 [==============================] - 5s 89us/step - loss: 1.4848 - accuracy: 0.3420
- val_loss: 1.4816 - val_accuracy: 0.3532
Epoch 9/10
60000/60000 [==============================] - 5s 90us/step - loss: 1.4520 - accuracy: 0.3528
- val_loss: 1.4475 - val_accuracy: 0.3702
Epoch 10/10
60000/60000 [==============================] - 5s 91us/step - loss: 1.4244 - accuracy: 0.3711
- val_loss: 1.4248 - val_accuracy: 0.3764
```

Conv2DTranspose

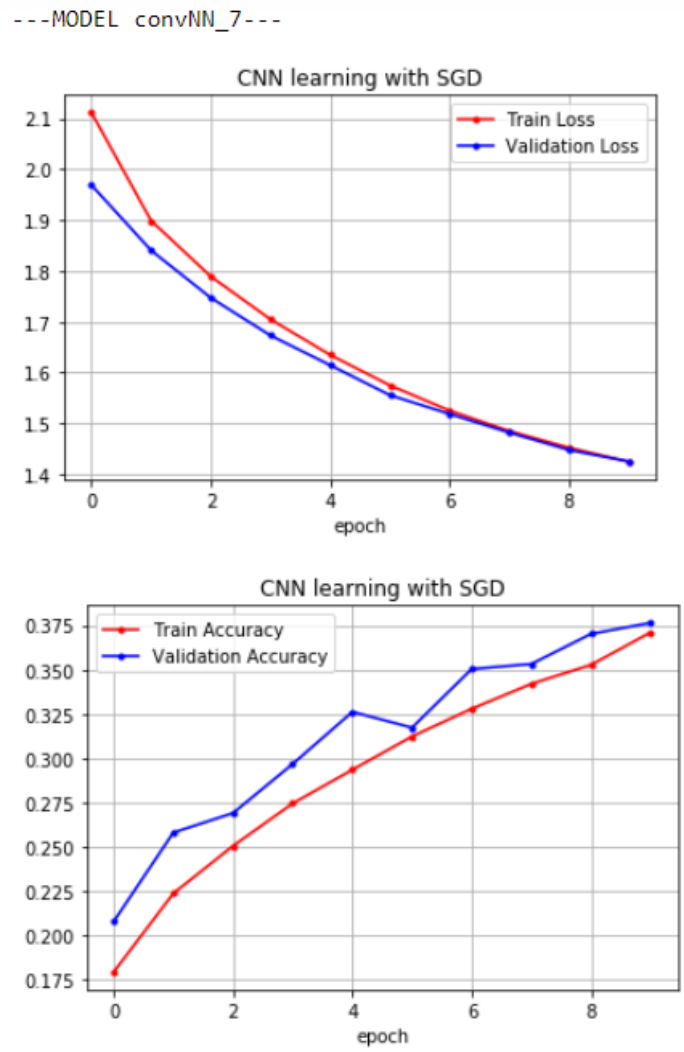
```
 Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 40s 667us/step - loss: 1.0653 - accuracy: 0.6532
- val_loss: 0.6588 - val_accuracy: 0.7532
Epoch 2/10
60000/60000 [==============================] - 41s 681us/step - loss: 0.6137 - accuracy: 0.7773
- val_loss: 0.6355 - val_accuracy: 0.7811
Epoch 3/10
```

```
60000/60000 [==============================] - 42s 695us/step - loss: 0.5353 - accuracy: 0.8068
- val_loss: 0.5195 - val_accuracy: 0.8168
Epoch 4/10
60000/60000 [==============================] - 42s 699us/step - loss: 0.4914 - accuracy: 0.8238
- val_loss: 0.4968 - val_accuracy: 0.8126
Epoch 5/10
60000/60000 [==============================] - 42s 704us/step - loss: 0.4601 - accuracy: 0.8374
- val_loss: 0.4589 - val_accuracy: 0.8353
Epoch 6/10
60000/60000 [==============================] - 43s 724us/step - loss: 0.4432 - accuracy: 0.8412
- val_loss: 0.4907 - val_accuracy: 0.8213
Epoch 7/10
60000/60000 [==============================] - 45s 744us/step - loss: 0.4218 - accuracy: 0.8511
- val_loss: 0.4624 - val_accuracy: 0.8329
Epoch 8/10
60000/60000 [==============================] - 44s 734us/step - loss: 0.4055 - accuracy: 0.8551
- val_loss: 0.4084 - val_accuracy: 0.8540
Epoch 9/10
60000/60000 [==============================] - 44s 739us/step - loss: 0.3889 - accuracy: 0.8622
- val_loss: 0.4069 - val_accuracy: 0.8564
Epoch 10/10
60000/60000 [==============================] - 44s 728us/step - loss: 0.3781 - accuracy: 0.8654
- val_loss: 0.3912 - val_accuracy: 0.8603
```

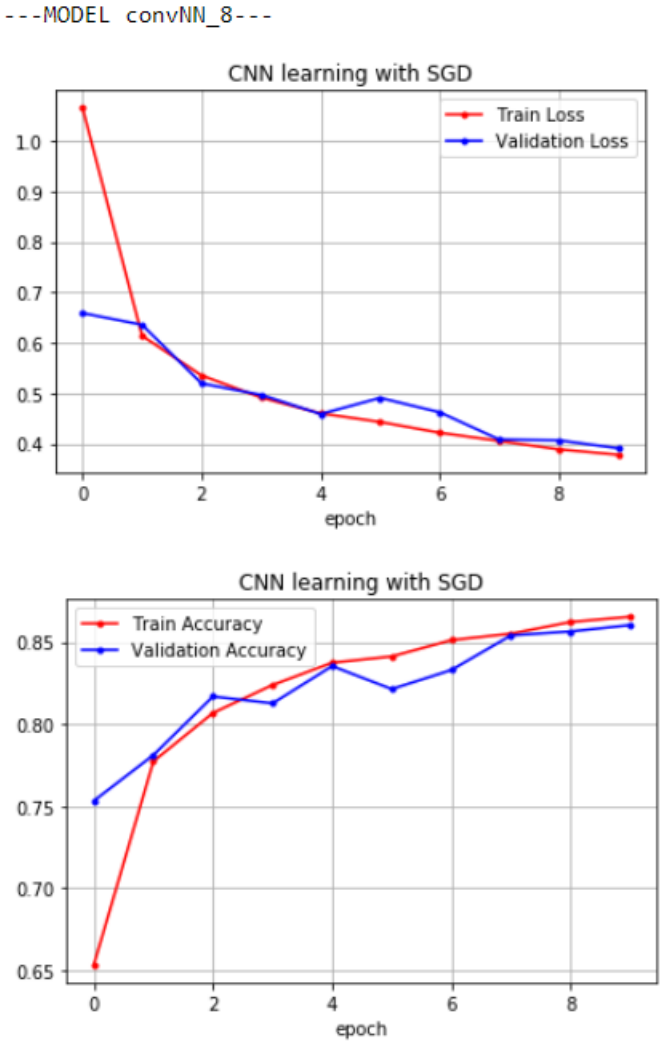
```
10000/10000 [==============================] - 1s 84us/step
Test 1 accuracy: 0.8425999879837036
10000/10000 [==============================] - 1s 77us/step
Test 2 accuracy: 0.8443999886512756
10000/10000 [==============================] - 2s 151us/step
Test 3 accuracy: 0.8392999768257141
10000/10000 [==============================] - 2s 162us/step
Test 3a accuracy: 0.8511000275611877
10000/10000 [==============================] - 2s 217us/step
Test 4 accuracy: 0.8521999716758728
10000/10000 [==============================] - 1s 133us/step
Test 6 accuracy: 0.8525999784469604
10000/10000 [==============================] - 1s 54us/step
Test 7 accuracy: 0.3763999938964844
10000/10000 [==============================] - 3s 294us/step
Test 8 accuracy: 0.8603000044822693
```



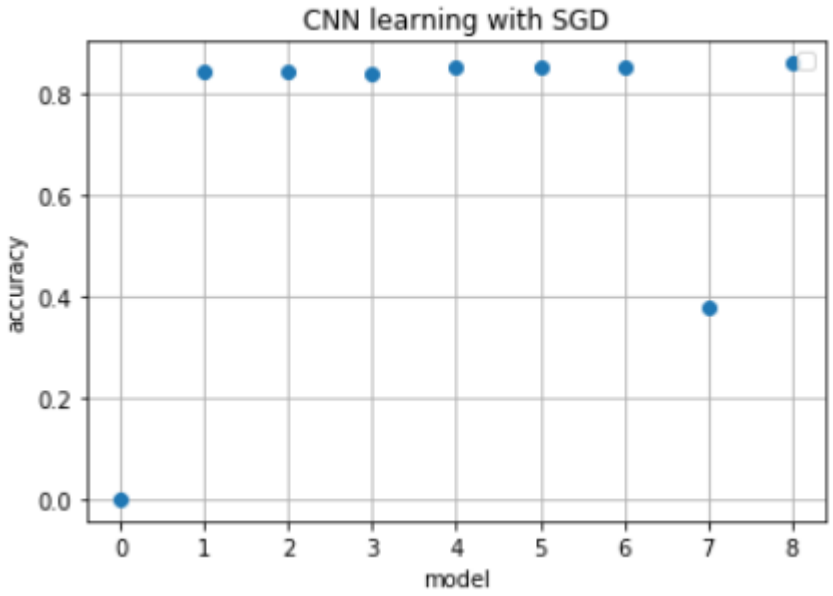
Rysunek 3.14: *Źródło: własne*



Rysunek 3.15: *Źródło: własne*



Rysunek 3.16: *Źródło: własne*



3.4 Wpływ parametrów konfiguracyjnych

Grid search

Przeszukiwaniem parametrów GridSearch można w prosty i wygodny sposób znaleźć parametry, które powiny być bardziej optymalne. Po przeprowadzeniu kilku prób odnalezienia najlepszych parametrów otrzymaliśmy:

```
 Best model :
{   'activation': 'relu',
    'batch_size': 10,
    'dropout_rate': 0,
    'epochs': 15,
    'initializer': 'lecun_uniform',
    'learning_rate': 0.1,
    'num_unit': 500,
    'optimizer': <class 'keras.optimizers.SGD'>}
```

Z wynikiem:

```
60000/60000 [==============================] - 34s 571us/step
- loss: 0.2061 - accuracy: 0.9209
```

Nie udało am się uzyskać lepszego wyniku, jeśli chodzi o sprawdzany model:

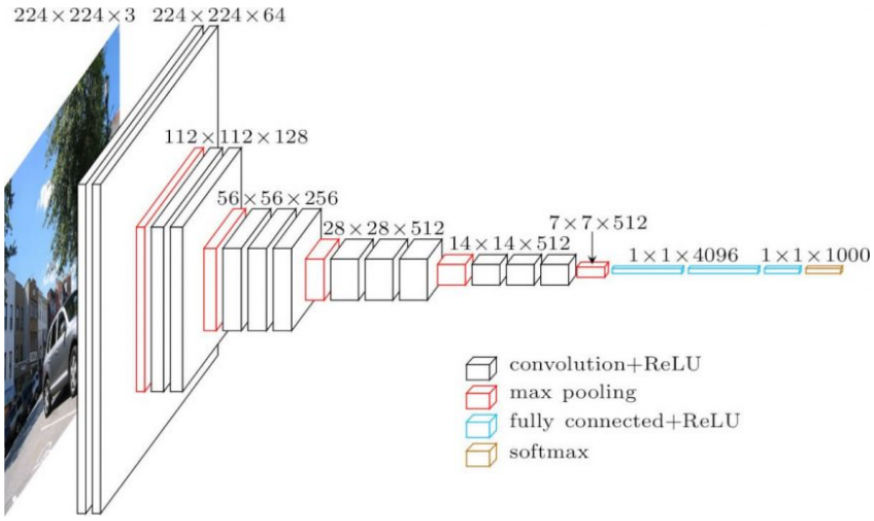
```
def build_model(optimizer, learning_rate, activation, dropout_rate,
initializer, num_unit):


    model = Sequential()
    model.add(Flatten())
    model.add(Dense(num_unit, kernel_initializer=initializer,
    activation=activation, input_shape=(28, 28, 1)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(num_unit, kernel_initializer=initializer,
    activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(num_unit, kernel_initializer=initializer,
    activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
    optimizer=optimizer(lr=learning_rate), metrics=['accuracy'])
    return model
```

3.5 Sieć VGG16

VGG16 jest modelem splotowej sieci neuronowej zaproponowanym przez K. Simonyana i A. Zissermana z University of Oxford w artykule „Very Deep Convolutional Networks for Large Scale Image Recognition". Model osiąga 92,7 procent dokładności testu w pierwszej piątce w ImageNet, który jest zbiorem danych ponad 14 milionów obrazów należących do 1000 klas. Wejście do warstwy cov1 ma stały rozmiar obrazu RGB 224 x 224. Obraz jest przepuszczany przez stos warstw splotowych (konw.), W których zastosowano filtry z bardzo małym polem recepcyjnym: 3 3. Architektura została przedstawiona na rysunku 2.18.

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 48, 48, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 48, 48, 64)        1792
```



Rysunek 3.17: *Źródło: neurohive.io*

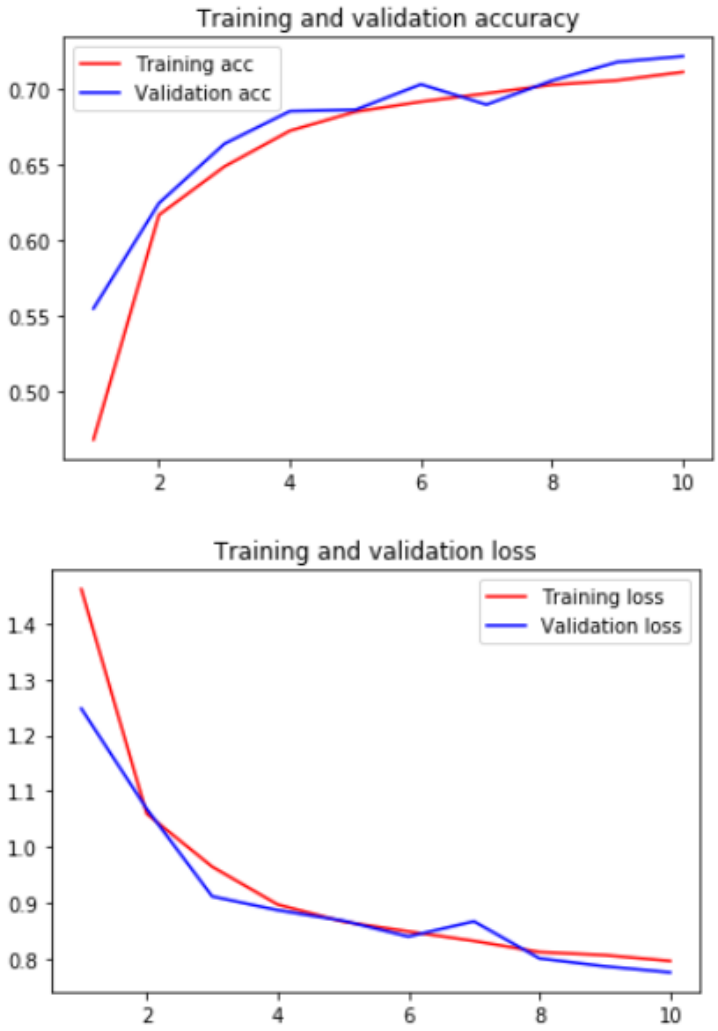
block1_conv2 (Conv2D)	(None, 48, 48, 64)	36928
block1_pool (MaxPooling2D)	(None, 24, 24, 64)	0
block2_conv1 (Conv2D)	(None, 24, 24, 128)	73856
block2_conv2 (Conv2D)	(None, 24, 24, 128)	147584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
block4_conv1 (Conv2D)	(None, 6, 6, 512)	1180160
block4_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block4_conv3 (Conv2D)	(None, 6, 6, 512)	2359808

```
block4_pool (MaxPooling2D)   (None, 3, 3, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Udało nam się osiągnąć następujące wyniki:

```
    Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 6s 126us/step
- loss: 1.4608 - acc: 0.4679 - val_loss: 1.2474 - val_acc: 0.5545
Epoch 2/10
48000/48000 [==============================] - 6s 120us/step
- loss: 1.0595 - acc: 0.6163 - val_loss: 1.0682 - val_acc: 0.6241
Epoch 3/10
48000/48000 [==============================] - 6s 119us/step
- loss: 0.9650 - acc: 0.6483 - val_loss: 0.9120 - val_acc: 0.6633
Epoch 4/10
48000/48000 [==============================] - 6s 120us/step
- loss: 0.8975 - acc: 0.6719 - val_loss: 0.8874 - val_acc: 0.6848
Epoch 5/10
48000/48000 [==============================] - 6s 123us/step
- loss: 0.8669 - acc: 0.6845 - val_loss: 0.8686 - val_acc: 0.6858
Epoch 6/10
48000/48000 [==============================] - 6s 127us/step
- loss: 0.8493 - acc: 0.6911 - val_loss: 0.8402 - val_acc: 0.7026
Epoch 7/10
48000/48000 [==============================] - 6s 127us/step
- loss: 0.8325 - acc: 0.6965 - val_loss: 0.8674 - val_acc: 0.6892
Epoch 8/10
48000/48000 [==============================] - 6s 129us/step
- loss: 0.8126 - acc: 0.7023 - val_loss: 0.8013 - val_acc: 0.7050
Epoch 9/10
```

```
48000/48000 [==============================] - 6s 129us/step
- loss: 0.8071 - acc: 0.7052 - val_loss: 0.7870 - val_acc: 0.7173
Epoch 10/10
48000/48000 [==============================] - 6s 129us/step
- loss: 0.7966 - acc: 0.7107 - val_loss: 0.7763 - val_acc: 0.7211
```



Rysunek 3.18: *Źródło: własne*

Rozdział 4

Podsumowanie

Głównym problemem przy trenowaniu modeli sieci splotowych jest dobór odpowiednich parametrów modelu. Narzędzi atakie jak GridSearch czy Cross Validation są nieocenione przy tym procesie. Należy jednak pamiętać, że uczenie maszynowe opiera się na wyczuciu i intuicji, a przede wszystkim wymaga ogromnej ilości czasu. Warto również wiedzieć, kiedy należy przestać szukać i uznać, że odnalezione przez nas parametry są wystarczająco dobre, a co ważniejsze czas wytrenowania modelu nie jest zbyt długi. Mozna zauważyc, że często stosunek poprawy jakości modelu (mierzonej różnymi metrykami m.in. accuracy) do zwiększenia czasu trzenowania jest zdecydowanie nieopłacalny. Często rownież nie odczujemy aż takiej różnicy iędzy wystarczająco dobrym, a jesczze lepszym modelem. Stąd warto pamiętać, że budowanie i treowanie sieci splotowych jak i całe uczenie maszyowe to próby, próby i błędy, a z nich wnioski. Tego przede wszystkim dowodem był ninijeszy projekt. Mozna więc powiedzieć, że sukces gwarantują odpowiednie i odpowiednio przygotowane dane wejsściowe, dobrze dobrane parametry modelu trenującego oraz prawidłowo zinterpretowane wyniki pomiarów odpowiednimi metrykami, na których podstawie (biorąc pod uwagę nasze potrzeby, wymagania co do modelu i ograniczenia) można tworzyć coraz lepsze rozwiązania.

Bibliografia

[1] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems*, 2017.

[2] Brian Lao, Karthik Jagadeesh, "Convolutional Neural Networks for Fashion Classification and Object Detection " [online] http://cs231n.stanford.edu/reports/2015/pdfs/ BLAO KJAG CS231N FinalPaperFashionClassification.pdf [dostęp: 16.06.2020]

[3] Krzysztof Wołk, "Konwolucyjne sieci neuronowe" [online] https://pclab.pl/art79689-3.html [dostęp: 16.06.2020]

[4] Margaret Maynard-Reid, "Fashion-MNIST with tf.Keras" [online] https://medium.com/tensorflow/ hello-deep-learning-fashion- mnist-with-keras-50fcff8cd74a [dostęp: 16.06.2020]

[5] "VGG16 – Convolutional Network for Classification and Detection" [online] https://neurohive.io/en/popular-networks /vgg16/ [dostęp: 16.06.2020]

[6] Karol Piczak, *Rozprawa doktorska. Klasyfikacja dźwięku za pomocą splotowych sieci neuronowych*, Warszawa, 2018.

[7] Tensorflow Documentation, [online], https://www.tensorflow.org

[8] kaggle.com

Dodatek A

Kod programu

https://github.com/kelament/SI-2019-2020-Projekt-Maria-Guz-Karol-P-tko/blob/master/kod.py

```
import keras
from keras.datasets import fashion_mnist

(data_train, target_train), (data_test, target_test) = fashion_mnist.load_data()

print('data_train:', data_train.shape)
print(target_train)
print('data_test:', data_test.shape)
print(target_test)

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


import matplotlib.pyplot as plt
%matplotlib inline
for i in range(0,10):
    class_name = class_names[target_train[i]]
    plt.figure(figsize=(3,3))
    plt.imshow(data_train[i])
    plt.axis('off')
    plt.colorbar()
    plt.title(class_name)



num_classes = 10

target_train = keras.utils.to_categorical(target_train, num_classes)
target_test = keras.utils.to_categorical(target_test, num_classes)

print(target_train[5])



data_train = data_train.astype('float32')
data_test = data_test.astype('float32')

#normalize data
data_train /= 255.0
data_test /= 255.0
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, SeparableConv2D, DepthwiseConv2D, Conv2DTranspose

data_train=data_train.reshape(data_train.shape[0], *(28,28,1))
data_test=data_test.reshape(data_test.shape[0], *(28,28,1))


### bez dropoutów



convNN_1 = Sequential()
convNN_1.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_1.add(MaxPooling2D((2, 2)))
convNN_1.add(Flatten())
convNN_1.add(Dense(100))
convNN_1.add(Activation('relu'))
convNN_1.add(Dense(10))
convNN_1.add(Activation('softmax'))
convNN_1.summary()

convNN_2 = Sequential()
convNN_2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_2.add(MaxPooling2D((2, 2)))
convNN_2.add(Dropout(0.15))
convNN_2.add(Flatten())
convNN_2.add(Dense(100))
convNN_2.add(Activation('relu'))
convNN_2.add(Dense(10))
convNN_2.add(Activation('softmax'))
convNN_2.summary()


convNN_3 = Sequential()
convNN_3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_3.add(MaxPooling2D((2, 2)))
# convNN_3.add(Dropout(0.1))
convNN_3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
# convNN_3.add(Dropout(0.1))
convNN_3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
# convNN_3.add(Dropout(0.1))
convNN_3.add(Flatten())
convNN_3.add(Dense(100))
convNN_3.add(Activation('relu'))
convNN_3.add(Dense(10))
convNN_3.add(Activation('softmax'))
convNN_3.summary()


convNN_3a = Sequential()
convNN_3a.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_3a.add(MaxPooling2D((2, 2)))
convNN_3a.add(Dropout(0.1))
convNN_3a.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_3a.add(Dropout(0.1))
convNN_3a.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_3a.add(Dropout(0.1))
convNN_3a.add(Flatten())
convNN_3a.add(Dense(100))
convNN_3a.add(Activation('relu'))
```

```
convNN_3a.add(Dense(10))
convNN_3a.add(Activation('softmax'))
convNN_3a.summary()


convNN_4 = Sequential()
convNN_4.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_4.add(MaxPooling2D((2, 2)))
convNN_4.add(Dropout(0.1))
convNN_4.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_4.add(Dropout(0.1))
convNN_4.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_4.add(Dropout(0.1))
convNN_4.add(Flatten())
convNN_4.add(Dense(1000))
convNN_4.add(Activation('relu'))
convNN_4.add(Dense(500))
convNN_4.add(Activation('relu'))
convNN_4.add(Dense(100))
convNN_4.add(Activation('relu'))
convNN_4.add(Dense(10))
convNN_4.add(Activation('softmax'))
convNN_4.summary()

convNN_6 = Sequential()
convNN_6.add(SeparableConv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_6.add(MaxPooling2D(pool_size=2))
convNN_6.add(Flatten())
convNN_6.add(Dense(100))
convNN_6.add(Activation('relu'))
convNN_6.add(Dense(10))
convNN_6.add(Activation('softmax'))
convNN_6.summary()


convNN_7 = Sequential()
convNN_7.add(DepthwiseConv2D(28, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_7.add(Flatten())
convNN_7.add(Dense(100))
convNN_7.add(Activation('relu'))
convNN_7.add(Dense(10))
convNN_7.add(Activation('softmax'))
convNN_7.summary()



convNN_8 = Sequential()
convNN_8.add(Conv2DTranspose(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
convNN_8.add(Activation('relu'))
convNN_8.add(MaxPooling2D(pool_size=2))
convNN_8.add(Flatten())
convNN_8.add(Dense(100))
convNN_8.add(Activation('relu'))
convNN_8.add(Dense(10))
convNN_8.add(Activation('softmax'))
convNN_8.summary()


#to plot model as png file
from keras.utils import plot_model
model_name="convNN_1"
plot_model(convNN_1, to_file=model_name+'.png')
```

```
#to plot model as png file
from keras.utils import plot_model
model_name="convNN_2"
plot_model(convNN_2, to_file=model_name+'.png')
#to plot model as png file
from keras.utils import plot_model
model_name="convNN_3"
plot_model(convNN_3, to_file=model_name+'.png')
#to plot model as png file
from keras.utils import plot_model
model_name="convNN_3a"
plot_model(convNN_3a, to_file=model_name+'.png')
#to plot model as png file
from keras.utils import plot_model
model_name="convNN_4"
plot_model(convNN_4, to_file=model_name+'.png')
#to plot model as png file
from keras.utils import plot_model
model_name="convNN_6"
plot_model(convNN_6, to_file=model_name+'.png')

#to plot model as png file
from keras.utils import plot_model
model_name="convNN_7"
plot_model(convNN_7, to_file=model_name+'.png')

#to plot model as png file
from keras.utils import plot_model
model_name="convNN_8"
plot_model(convNN_8, to_file=model_name+'.png')

#hyberparameters parameters
batch_size = 500
epochs = 10

# select and initiate optimizer
opt_sgd = keras.optimizers.sgd(lr=0.05)

convNN_1.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd = convNN_1.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(data

convNN_2.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_2 = convNN_2.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(da

convNN_3.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_3 = convNN_3.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(da

convNN_3a.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_3a = convNN_3a.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(

convNN_4.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_4 = convNN_4.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(da

convNN_6.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_6 = convNN_6.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(da
```

```
convNN_7.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_7 = convNN_7.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(da

convNN_8.compile(loss='categorical_crossentropy', optimizer=opt_sgd, metrics=['accuracy'])

run_hist_sgd_8 = convNN_8.fit(data_train, target_train, batch_size=batch_size, epochs=epochs, validation_data=(da


print("---MODEL convNN_1---")

plt.plot(run_hist_sgd.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()

plt.plot(run_hist_sgd.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()


print("---MODEL convNN_2---")

plt.plot(run_hist_sgd_2.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_2.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()

plt.plot(run_hist_sgd_2.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_2.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()


print("---MODEL convNN_3---")

plt.plot(run_hist_sgd_3.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_3.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()

plt.plot(run_hist_sgd_3.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_3.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
```

```
print("---MODEL convNN_3a---")

plt.plot(run_hist_sgd_3a.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_3a.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()

plt.plot(run_hist_sgd_3a.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_3a.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()


print("---MODEL convNN_4---")

plt.plot(run_hist_sgd_4.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_4.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()

plt.plot(run_hist_sgd_4.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_4.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()


print("---MODEL convNN_6---")

plt.plot(run_hist_sgd_6.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_6.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()


plt.plot(run_hist_sgd_6.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_6.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()


print("---MODEL convNN_7---")

plt.plot(run_hist_sgd_7.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_7.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
```

```
plt.legend()
plt.grid()
plt.show()


plt.plot(run_hist_sgd_7.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_7.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()


print("---MODEL convNN_8---")

plt.plot(run_hist_sgd_8.history["loss"], 'r', marker='.', label="Train Loss")
plt.plot(run_hist_sgd_8.history["val_loss"], 'b', marker='.', label="Validation Loss")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()
plt.show()


plt.plot(run_hist_sgd_8.history["accuracy"], 'r', marker='.', label="Train Accuracy")
plt.plot(run_hist_sgd_8.history["val_accuracy"], 'b', marker='.', label="Validation Accuracy")
plt.xlabel("epoch")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()

#model accuracy
scores1 = convNN_1.evaluate(data_test, target_test, verbose=True)
print('Test 1 accuracy:', scores1[1])

scores2 = convNN_2.evaluate(data_test, target_test, verbose=True)
print('Test 2 accuracy:', scores2[1])

scores3 = convNN_3.evaluate(data_test, target_test, verbose=True)
print('Test 3 accuracy:', scores3[1])

scores3a = convNN_3a.evaluate(data_test, target_test, verbose=True)
print('Test 3a accuracy:', scores3a[1])

scores4 = convNN_4.evaluate(data_test, target_test, verbose=True)
print('Test 4 accuracy:', scores4[1])


scores6 = convNN_6.evaluate(data_test, target_test, verbose=True)
print('Test 6 accuracy:', scores6[1])

scores7 = convNN_7.evaluate(data_test, target_test, verbose=True)
print('Test 7 accuracy:', scores7[1])

scores8 = convNN_8.evaluate(data_test, target_test, verbose=True)
print('Test 8 accuracy:', scores8[1])


scores_plot = [0, scores1[1], scores2[1], scores3[1], scores3a[1], scores4[1], scores6[1], scores7[1], scores8[1]

plt.plot(scores_plot, "o")
plt.xlabel("model")
```

```
plt.ylabel("accuracy")
plt.title("CNN learning with SGD")
plt.legend()
plt.grid()



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import *
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import to_categorical
from sklearn.model_selection import GridSearchCV
import pprint
pp = pprint.PrettyPrinter(indent = 4)

def build_model(optimizer, learning_rate, activation, dropout_rate, initializer, num_unit):


    model = Sequential()
    model.add(Flatten())
    model.add(Dense(num_unit, kernel_initializer=initializer, activation=activation, input_shape=(28, 28, 1)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(num_unit, kernel_initializer=initializer, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(num_unit, kernel_initializer=initializer, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer(lr=learning_rate), metrics=['accuracy'])
    return model


batch_size = [500][:1]
epochs = [10][:1]
initializer = ['lecun_uniform'][:1]
learning_rate = [0.1][:1]
dropout_rate = [0, 0.1, 0.2, 0.3][:1]
num_unit = [1000, 3000 ][:1]
activation = ['relu', 'sigmoid'][:1]
optimizer = [SGD][:1]



parameters = dict(batch_size = batch_size,
                  epochs = epochs,
                  dropout_rate = dropout_rate,
                  num_unit = num_unit,
                  initializer = initializer,
                  learning_rate = learning_rate,
                  activation = activation,
                  optimizer = optimizer)



model = KerasClassifier(build_fn=build_model, verbose=1)
models = GridSearchCV(estimator = model, param_grid=parameters, n_jobs=1)
```

```
best_model = models.fit(data_train, target_train)
print('Best model :')
pp.pprint(best_model.best_params_)


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import *
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import to_categorical
from sklearn.model_selection import GridSearchCV
import pprint
pp = pprint.PrettyPrinter(indent = 4)

def build_model(optimizer, learning_rate, activation, dropout_rate, initializer, num_unit):


    model = Sequential()
    model.add(Flatten())
    model.add(Dense(num_unit, kernel_initializer=initializer, activation=activation, input_shape=(28, 28, 1)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(num_unit, kernel_initializer=initializer, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(num_unit, kernel_initializer=initializer, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer(lr=learning_rate), metrics=['accuracy'])
    return model


batch_size = [10, 50, 100][:1]
epochs = [15][:1]
initializer = ['lecun_uniform'][:1]
learning_rate = [0.1][:1]
dropout_rate = [0][:1]
num_unit = [500][:1]
activation = ['relu'][:1]
optimizer = [SGD][:1]



parameters = dict(batch_size = batch_size,
                  epochs = epochs,
                  dropout_rate = dropout_rate,
                  num_unit = num_unit,
                  initializer = initializer,
                  learning_rate = learning_rate,
                  activation = activation,
                  optimizer = optimizer)



model = KerasClassifier(build_fn=build_model, verbose=1)
models = GridSearchCV(estimator = model, param_grid=parameters, n_jobs=1)


best_model = models.fit(data_train, target_train)
print('Best model :')
```

```
pp.pprint(best_model.best_params_)
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os, time
import matplotlib.pyplot as plt
#from keras.datasets import fashion_mnist
from sklearn.model_selection import train_test_split
import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Flatten
#from keras.layers.advanced_activations import LeakyReLU
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16;
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import img_to_array, array_to_img
import os
from keras import models
from keras.models import Model
from keras import layers
from keras import optimizers
from keras import callbacks
from keras.layers.advanced_activations import LeakyReLU
```

```
(data_train, target_train), (data_test, target_test) = fashion_mnist.load_data()
```

```
data_train=np.dstack([data_train] * 3)
data_test=np.dstack([data_test]*3)
data_train.shape
data_test.shape

data_train = data_train.reshape(-1, 28,28,3)
data_test= data_test.reshape (-1,28,28,3)
data_train.shape
data_test.shape
```

```
data_train = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in data_train])
data_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in data_test])
data_train.shape
data_test.shape

data_train = data_train / 255.
data_test = data_test / 255.
data_train = data_train.astype('float32')
data_test = data_test.astype('float32')
```

```
train_Y_one_hot = to_categorical(target_train)
test_Y_one_hot = to_categorical(target_test)
```

```
train_X,valid_X,train_label,valid_label = train_test_split(data_train, train_Y_one_hot, test_size=0.2, random_sta
```

```
IMG_WIDTH = 48
IMG_HEIGHT = 48
IMG_DEPTH = 3
BATCH_SIZE = 16

train_X = preprocess_input(train_X)
valid_X = preprocess_input(valid_X)
test_X  = preprocess_input (data_test)

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH)
                 )
conv_base.summary()

train_features = conv_base.predict(np.array(train_X), batch_size=BATCH_SIZE, verbose=1)
test_features = conv_base.predict(np.array(test_X), batch_size=BATCH_SIZE, verbose=1)
val_features = conv_base.predict(np.array(valid_X), batch_size=BATCH_SIZE, verbose=1)
np.savez("train_features", train_features, train_label)
np.savez("test_features", test_features, target_test)
np.savez("val_features", val_features, valid_label)

train_features_flat = np.reshape(train_features, (48000, 1*1*512))
test_features_flat = np.reshape(test_features, (10000, 1*1*512))
val_features_flat = np.reshape(val_features, (12000, 1*1*512))

NB_TRAIN_SAMPLES = train_features_flat.shape[0]
NB_VALIDATION_SAMPLES = val_features_flat.shape[0]
NB_EPOCHS = 10

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_dim=(1*1*512)))
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.Dense(num_classes, activation='softmax'))

model.compile(
    loss='categorical_crossentropy',
    optimizer=optimizers.Adam(),
    metrics=['acc'])


reduce_learning = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=2,
    verbose=1,
    mode='auto',
    epsilon=0.0001,
    cooldown=2,
    min_lr=0)

eary_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0,
    patience=7,
    verbose=1,
    mode='auto')

callbacks = [reduce_learning, eary_stopping]
```

```
history_vgg16 = model.fit(
    train_features_flat,
    train_label,
    epochs=NB_EPOCHS,
    validation_data=(val_features_flat, valid_label),
    callbacks=callbacks
)

acc = history_vgg16.history['acc']
val_acc = history_vgg16.history['val_acc']
loss = history_vgg16.history['loss']
val_loss = history_vgg16.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()



from __future__ import print_function
import keras
from keras.layers import Dense, Conv2D, BatchNormalization, Activation
from keras.layers import MaxPooling2D, AveragePooling2D, Input, Flatten
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
from keras.regularizers import l2
from keras import backend as K
from keras.models import Model
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt

(data_train, target_train), (data_test, target_test) = fashion_mnist.load_data()


%matplotlib inline
epochs = 5
batch_size = 100
data_augmentation = False
img_size = 28

num_classes = 10
num_filters = 64
num_blocks = 4
num_sub_blocks = 2
use_max_pool = False
```

```
x_train = data_train.reshape(data_train.shape[0],img_size,img_size,1)
x_test = data_test.reshape(data_test.shape[0],img_size,img_size,1)
input_size = (img_size, img_size,1)

# Normalize data.
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255



#Converting labels to one-hot vectors
y_train = keras.utils.to_categorical(target_train, num_classes)
y_test = keras.utils.to_categorical(target_test,num_classes)


inputs = Input(shape=input_size)
x = Conv2D(num_filters, padding='same',
           kernel_initializer='he_normal',
           kernel_size=7, strides=2,
           kernel_regularizer=l2(1e-4))(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)

if use_max_pool:
    x = MaxPooling2D(pool_size=3,padding='same', strides=2)(x)
    num_blocks =3


for i in range(num_blocks):
    for j in range(num_sub_blocks):
        strides = 1
        is_first_layer_but_not_first_block = j == 0 and i > 0


        if is_first_layer_but_not_first_block:
            strides = 2
        #Creating residual mapping using y
        y = Conv2D(num_filters,
                   kernel_size=3,
                   padding='same',
                   strides=strides,
                   kernel_initializer='he_normal',
                   kernel_regularizer=l2(1e-4))(x)
        y = BatchNormalization()(y)
        y = Activation('relu')(y)
        y = Conv2D(num_filters,
                   kernel_size=3,
                   padding='same',
                   kernel_initializer='he_normal',
                   kernel_regularizer=l2(1e-4))(y)
        y = BatchNormalization()(y)


        if is_first_layer_but_not_first_block:
            x = Conv2D(num_filters,
                       kernel_size=1,
                       padding='same',
                       strides=2,
                       kernel_initializer='he_normal',
                       kernel_regularizer=l2(1e-4))(x)
        x = keras.layers.add([x, y])
        x = Activation('relu')(x)
```

```
    num_filters = 2 * num_filters

x = AveragePooling2D()(x)
y = Flatten()(x)
outputs = Dense(num_classes,
                activation='softmax',
                kernel_initializer='he_normal')(y)

model = Model(inputs=inputs, outputs=outputs)
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
model.summary()


checkpoint = ModelCheckpoint(filepath=filepath,
                             verbose=1,
                             save_best_only=True)
lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               min_lr=0.5e-6)
callbacks = [checkpoint, lr_reducer]


if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True,
              callbacks=callbacks)


scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```