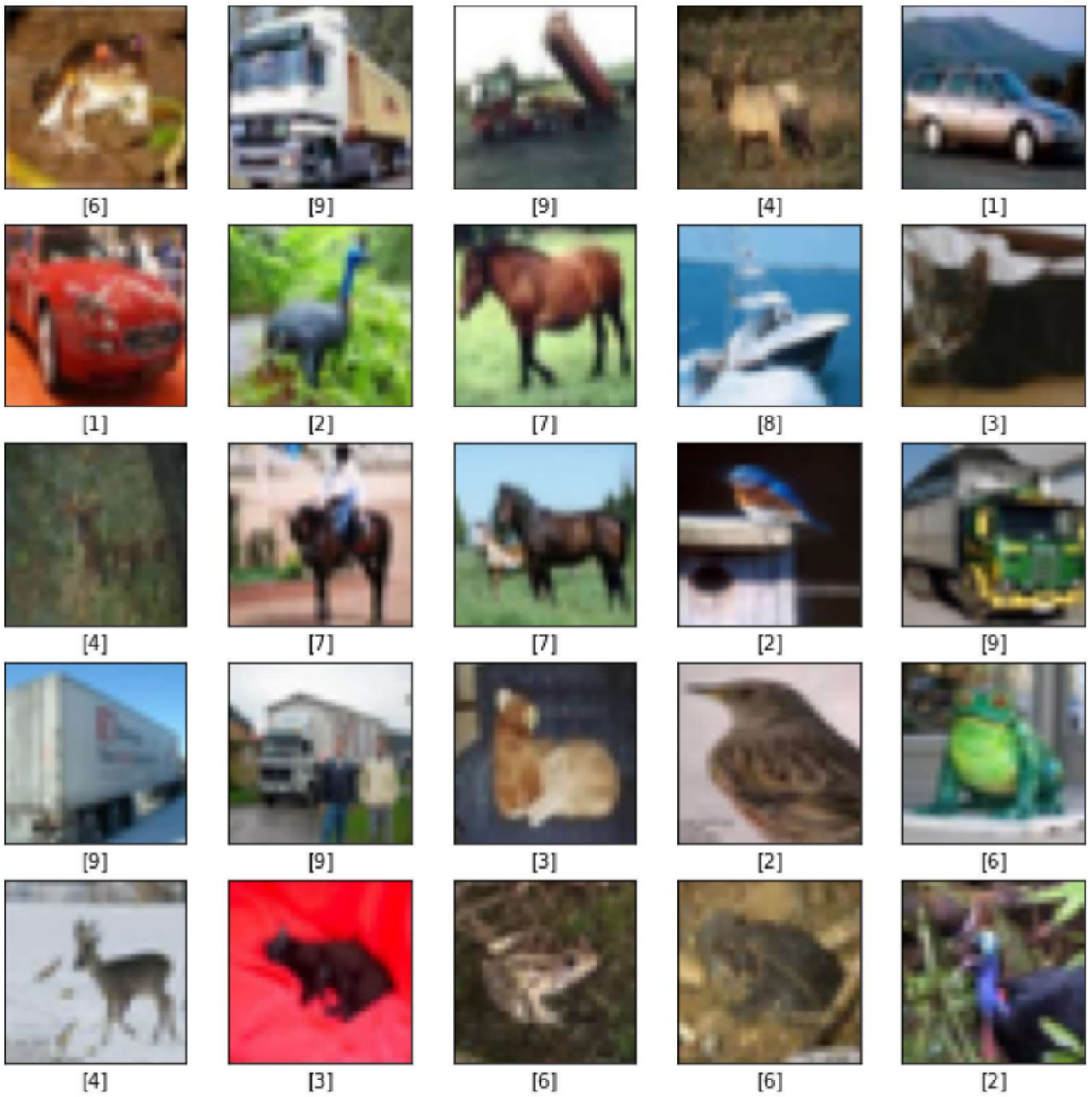
Metody Rozpoznawania Obrazów

Laboratorium 3

Maria Polak

1. Przygotowanie zbioru

Tensorflow oferuje wbudowaną możliwość wczytania zbioru CIFAR10. W zbiorze znajduje się 10 klas - samolot (0), samochód osobowy (1), ptak (2), kot (3), jeleń (4), pies (5), żaba (6), koń (7), statek (8), ciężarówka (9). W zbiorze znajduje się 60000 zdjęć (po 6 tysięcy na klasę), zdjęcia są róznorodne - obiekty są w różnych pozycjach, pod różnymi kątami i z innym oświetleniem.



Rysunek 1. Przykładowe elementy zbioru CIFAR10

Klasy 0-9 zostały przerobione na wektory prawdopodobieństw przy użyciu metody *to_categorical()* dostępnej w pakiecie tf.keras.utils.

```
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
```

```
[[6]
 [9]
 [9]
 ...
 [9]
 [1]
 [1]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]]
```

Klasy przed i po operacji to_categorical

2. Przygotowane modele

a. Mikro-architektura

Model został zaimplementowany w następujący sposób:

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=(32, 32, 3)))
model.add(tf.keras.layers.Rescaling(1 / 255))
model.add(tf.keras.layers.Conv2D(5, (3, 3), padding='same', activation='sigmoid'))
model.add(tf.keras.layers.Conv2D(5, (3, 3), padding='same', activation='sigmoid'))
model.add(tf.keras.layers.MaxPool2D(pool_size=(8, 8), strides=(8, 8)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.summary()
```

W podusmowaniu modelu widzimy, że w model zawiera 1180 parametrów, które należy dostroić.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_2 (Rescaling)      (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 5)         140
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 5)         230
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 5)           0
_________________________________________________________________
flatten_2 (Flatten)          (None, 80)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                810
=================================================================
Total params: 1,180
Trainable params: 1,180
Non-trainable params: 0
```

Podsumowanie modelu

Na wejście modelu zostało podane kilka zdjęć. Model zwrócił następujące wektory prawdopobieńst:

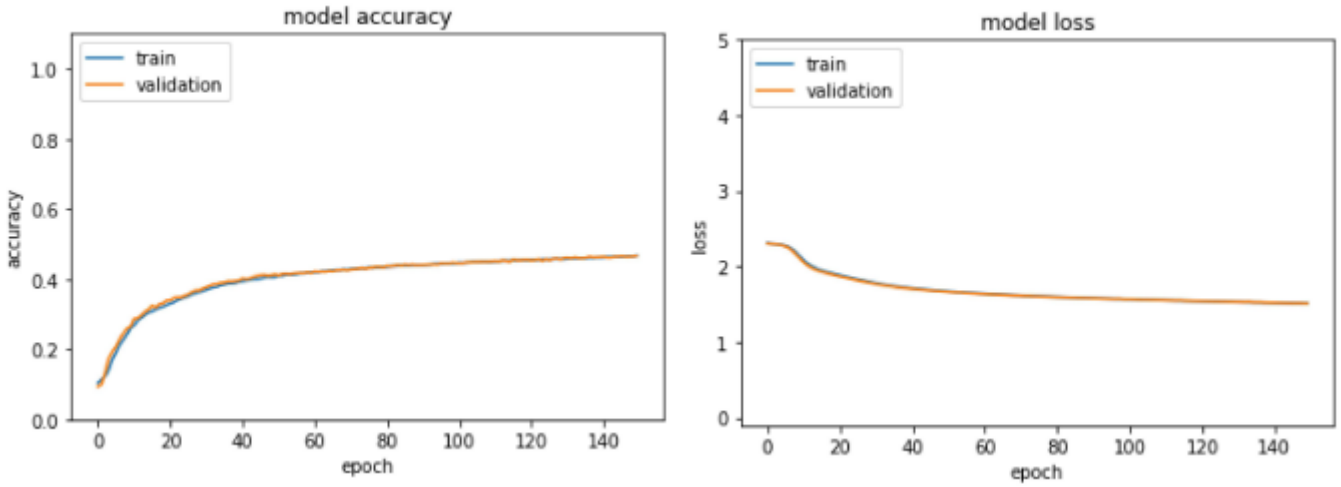
```
[[0.15223351 0.15054855 0.06711858 0.03272203 0.02132039 0.09141671
  0.17587979 0.09568182 0.04248472 0.17059384]
 [0.14904194 0.15452355 0.06816311 0.03127836 0.02270323 0.09200109
  0.17423931 0.09783833 0.04095195 0.16925919]
 [0.14975747 0.14990877 0.06952859 0.03155567 0.02282215 0.09237261
  0.1744951  0.09878423 0.04156712 0.1692083 ]
 [0.14993909 0.14915134 0.07004175 0.03169446 0.0224189  0.09270454
  0.1743451  0.09910399 0.04130352 0.16929737]
 [0.15278786 0.15133858 0.06659653 0.03160806 0.02138335 0.09208842
  0.17384095 0.09525906 0.04234531 0.17275187]
 [0.14976197 0.15159564 0.06857769 0.03249769 0.02184011 0.09198183
  0.17607224 0.09628639 0.04273979 0.16864659]...]
```

Możemy zauważyć, że niezależnie od zdjęcia na wejściu wyniki są bardzo zbliżone - każde odpowiednie prawdopodobieństwo klasy w kolejnych wektorach różni się o ok *0.01.*

Następnym krokiem było zoptymalizowanie modelu.

```
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9),
loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=x_prototype, y=y_prototype, batch_size=64, epochs=150,
                    validation_data=(x_test_prototype, y_test_prototype), verbose=0,
use_multiprocessing=True)
```

Średni czas wykonania 1 epoki treningu: 3,57 s



Wykres 1. Trendy optymalizacji pierwszego modelu

```
[[2.6484381e-02 2.7710995e-02 5.6908175e-02 3.0850205e-01 2.0662988e-02
  1.2569620e-01 3.0519509e-01 3.1763767e-03 1.1941027e-01 6.2535051e-03]
 [1.5867423e-01 5.3595471e-01 1.3197039e-03 4.9424019e-05 3.0567948e-04
  1.6675000e-05 2.2345741e-04 2.7605859e-04 1.6808069e-01 1.3509934e-01]
 [2.7355841e-01 4.6381470e-02 1.0231543e-02 5.2153533e-03 2.7048781e-03
  2.2777990e-03 9.1936433e-04 2.0340655e-03 6.2026393e-01 3.6413230e-02]
 [4.3116981e-01 5.1794171e-02 1.1993516e-01 3.3883300e-02 2.1511998e-02
  9.1667110e-03 5.9981118e-03 2.7682124e-02 2.0234121e-01 9.6517406e-02]
 [7.2664907e-03 2.0065075e-03 8.9652762e-02 1.6424327e-01 2.4895169e-01
  8.1111893e-02 3.7434873e-01 2.4021432e-02 7.0207119e-03 1.3764247e-03]
 [1.5294878e-02 2.6685882e-02 8.2623899e-02 1.8537244e-01 1.2836117e-01
  1.3726048e-01 2.1367918e-01 1.7759736e-01 8.0054421e-03 2.5119292e-02]...]
```

Tym razem prawdopodobieństwa różnią się między zdjęciami.

b. Więcej filtrów (5->20)

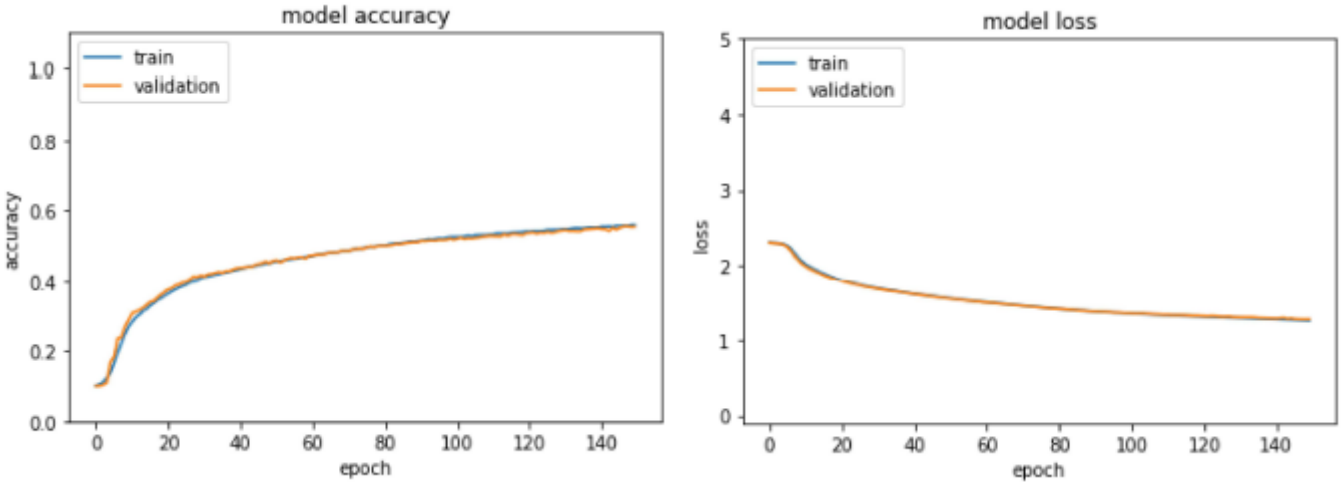
Na podstawie pierwszego modelu został stworzony kolejny z większą ilością filtrów w warstwach konwolucyjnych.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_3 (Rescaling)      (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 20)        560
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 20)        3620
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 20)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 320)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                3210
=================================================================
Total params: 7,390
Trainable params: 7,390
Non-trainable params: 0
```

Podsumowanie modelu

Liczba parametrów modelu wzrosła do 7390.

Średni czas wykonania 1 epoki treningu: 3,58 s (mimo kilkukrotnego zwiększenia się liczby parametrów, czas zmienił się minimalnie - 0,01s dłużej/epokę)



Wykres 2. Trendy optymalizacji modelu z większą ilością filtrów

c. Dwublokowy model - sigmoida

Zmieniony został rozmiar okna poolingu oraz przygotowana została funkcja dodająca blok konwolucyjny do przekazanego modelu.

```
def conv_block(filters, model):
```

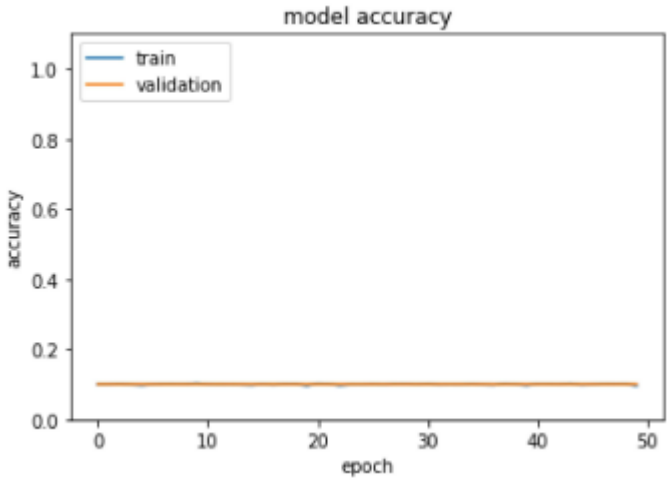
```
    model.add(tf.keras.layers.Conv2D(filters, (3, 3), padding='same',
activation='sigmoid'))
    model.add(tf.keras.layers.Conv2D(filters, (3, 3), padding='same',
activation='sigmoid'))
    model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Stworzony został kolejny modelu wykorzystujący dwa bloki konwolucyjne (20,40):

```
model3 = tf.keras.Sequential()
model3.add(tf.keras.layers.Input(shape=(32, 32, 3)))
model3.add(tf.keras.layers.Rescaling(1 / 255))

conv_block(20, model3)
conv_block(40, model3)

model3.add(tf.keras.layers.Flatten())
model3.add(tf.keras.layers.Dense(10, activation='softmax'))
model3.summary()
```



Wyniki trenowania na 50 epokach nie wyglądają obiecująco.

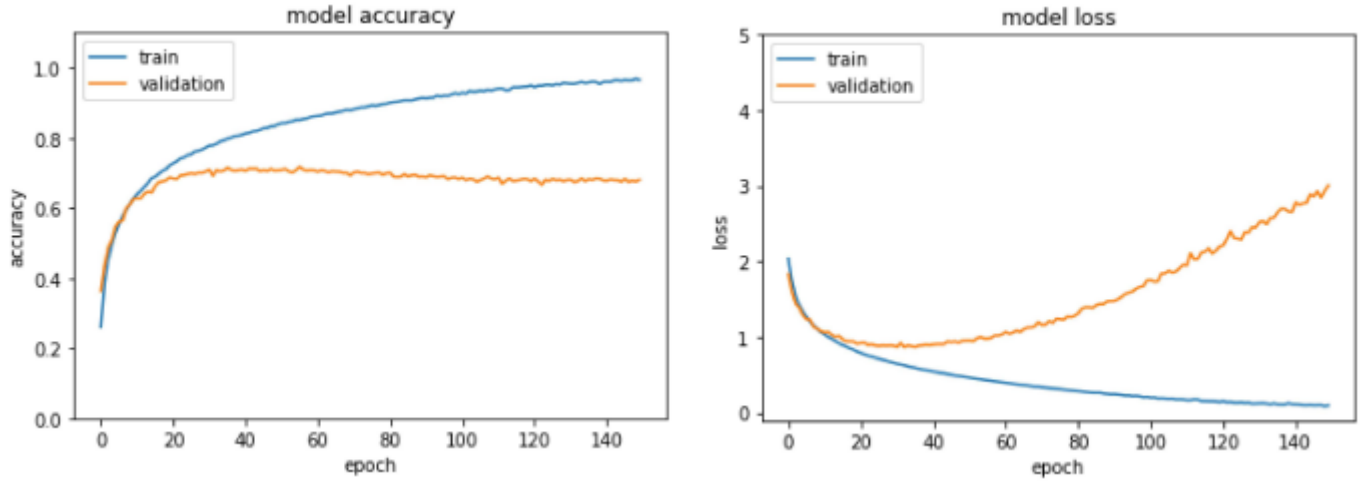
d. Dwublokowy model - ReLU

Zmodyfikowana została funkcja dodająca blok konwolucyjny, tak aby funkcją aktywującą było ReLU.

```
def conv_block_relu(filters, model):
    model.add(tf.keras.layers.Conv2D(filters, (3, 3), padding='same', activation='ReLU'))
    model.add(tf.keras.layers.Conv2D(filters, (3, 3), padding='same', activation='ReLU'))
    model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Przygotowany i wytrenowany został kolejny dwublokowy model korzystający z nowej funkcji.

Średni czas wykonania 1 epoki treningu: 4,65 s



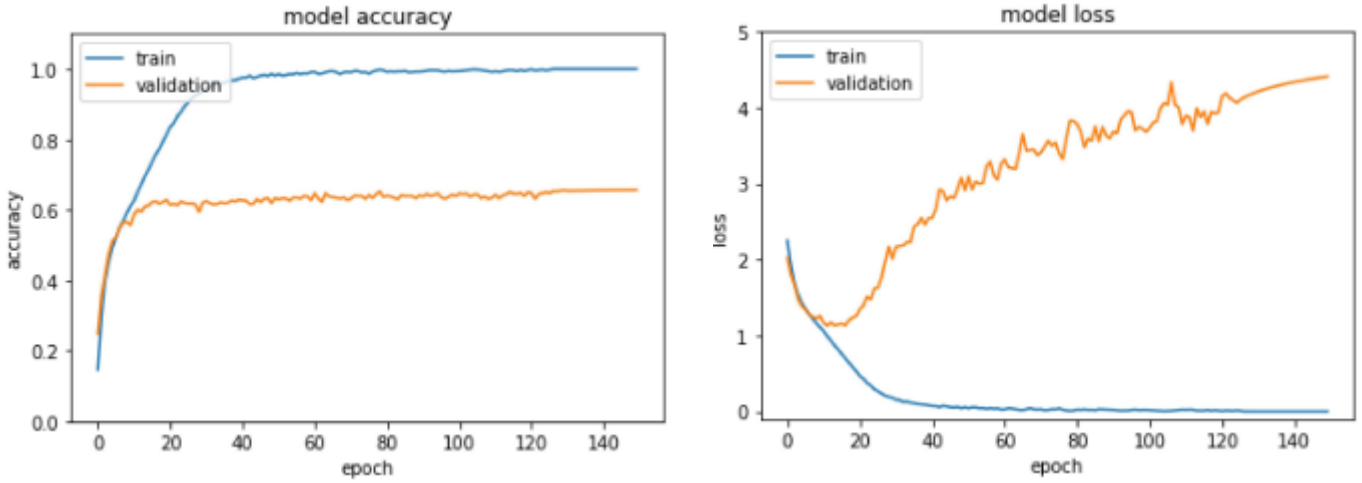
Wykres 3. Trendy optymalizacji modelu z dwoma blokami konwolucyjnymi i aktywacją ReLU

Możemy zauważyć, że w okolicach 30 epoki zaczął występować overfitting.

e. Czteroblokowy model

Do modelu dodane zostały kolejne dwa bloki konwolucyjne (80,160). Wyniki treningu są następujące:

Średni czas wykonania 1 epoki treningu: 6,40 s



Wykres 4. Trendy optymalizacji modelu z czterema blokami konwolucyjnymi

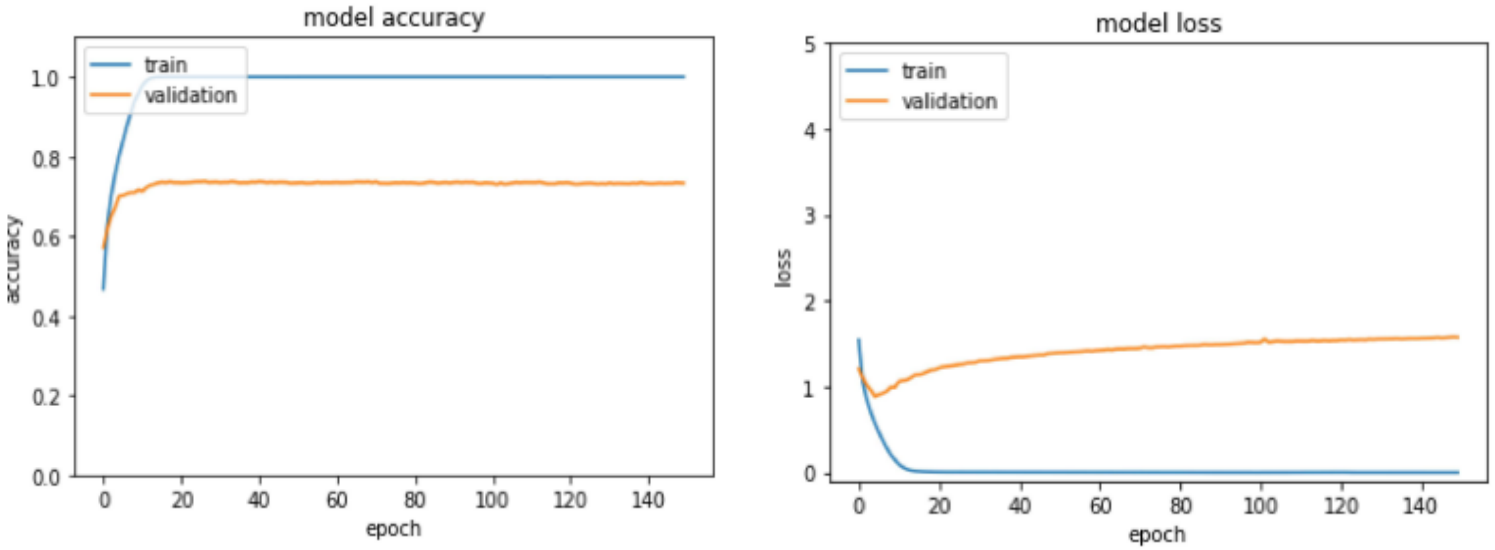
Na wykresie widać, że w okolicach 25 epoki celność na zbiorze walidacyjnym przestała wzrastać. Wyniki są porównywalne z poprzednim modelem.

f. Batch Normalization

Po każdej warstwie konwolucyjnej została dodana wartstwa “BatchNormalization”.

```
def conv_block_relu_batch(filters, model):
   model.add(tf.keras.layers.Conv2D(filters, (3, 3), padding='same', activation='ReLU'))
   model.add(tf.keras.layers.BatchNormalization())
   model.add(tf.keras.layers.Conv2D(filters, (3, 3), padding='same', activation='ReLU'))
   model.add(tf.keras.layers.BatchNormalization())
   model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Średni czas wykonania 1 epoki treningu: 8,04 s



Wykres 5. Trendy optymalizacji modelu z wartwami Batch Normalization

Na wykresach widać, że dodanie warstwy znacznie stablizuje proces uczenia. Wszystkie wykresy stały się dużo gładsze.

g. Dropout

Po każdej warstwie poolingowej dodana została warstwa “Dropout”

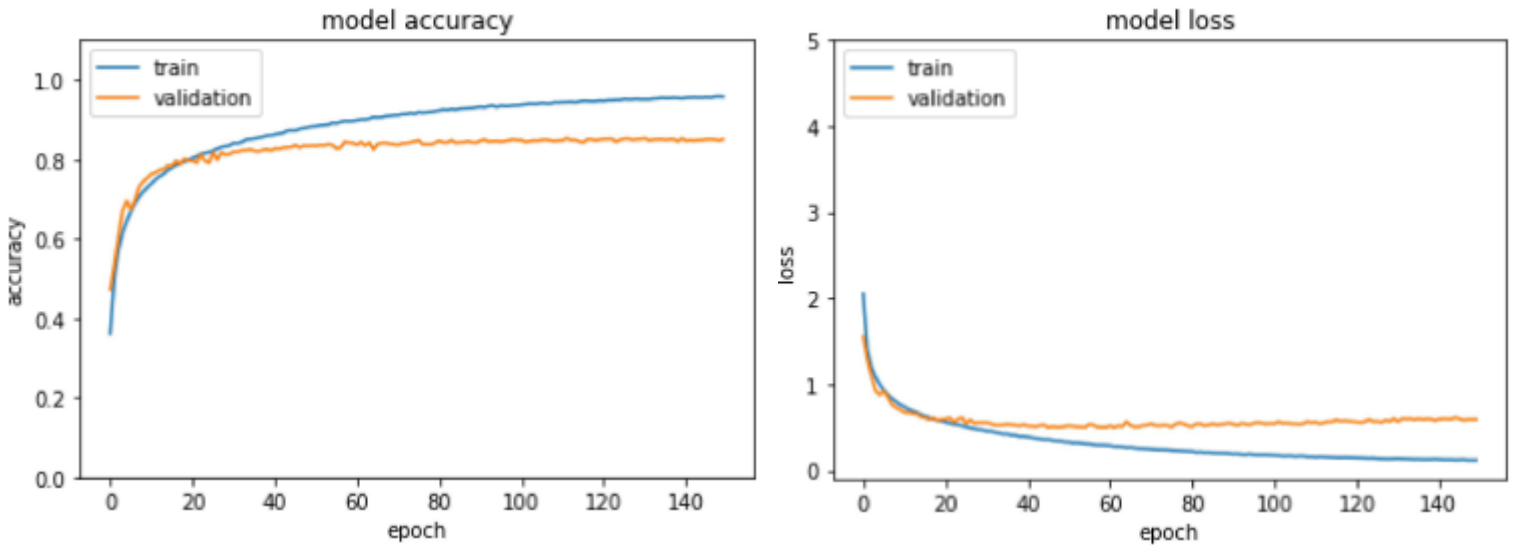
```
conv_block_relu_batch(20, model7)
model7.add(tf.keras.layers.Dropout(0.1))

conv_block_relu_batch(40, model7)
model7.add(tf.keras.layers.Dropout(0.2))

conv_block_relu_batch(80, model7)
model7.add(tf.keras.layers.Dropout(0.3))

conv_block_relu_batch(160, model7)
model7.add(tf.keras.layers.Dropout(0.4))
```

Średni czas wykonania 1 epoki treningu: 8,38 s



Wykres 6. Trendy optymalizacji modelu z wartwami Dropout

h. GAP

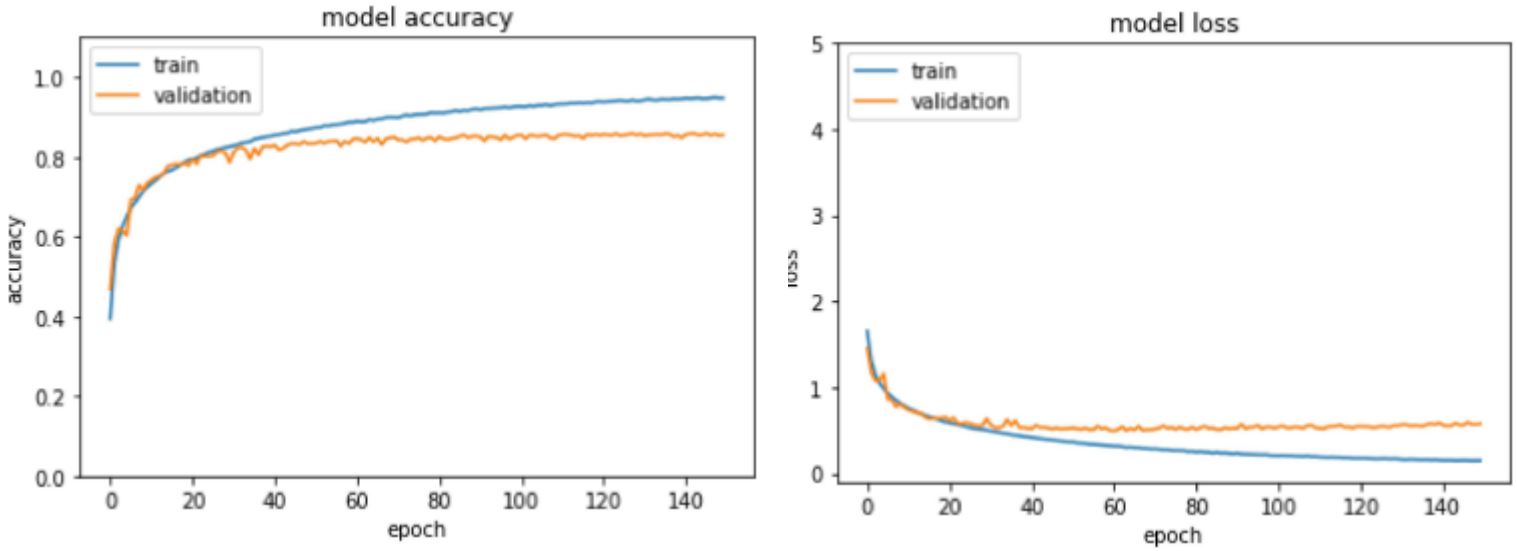
Wejście modelu zostało zmodyfikowane tak, aby przyjmowało obrazy o dowolnym rozmiarze.

```
model8.add(tf.keras.layers.Input(shape=(None, None, 3)))
```

Ostatni blok konwolucyjny został zmodfikowany tak, aby zawierał warstwę GAP zamiat MaxPooling

```
model8.add(tf.keras.layers.Conv2D(160, (3, 3), padding='same', activation='ReLU'))
model8.add(tf.keras.layers.BatchNormalization())
model8.add(tf.keras.layers.Conv2D(160, (3, 3), padding='same', activation='ReLU'))
model8.add(tf.keras.layers.BatchNormalization())
model8.add(tf.keras.layers.GlobalAvgPool2D())
model8.add(tf.keras.layers.Dropout(0.4))
```

Średni czas wykonania 1 epoki treningu: 8,12 s



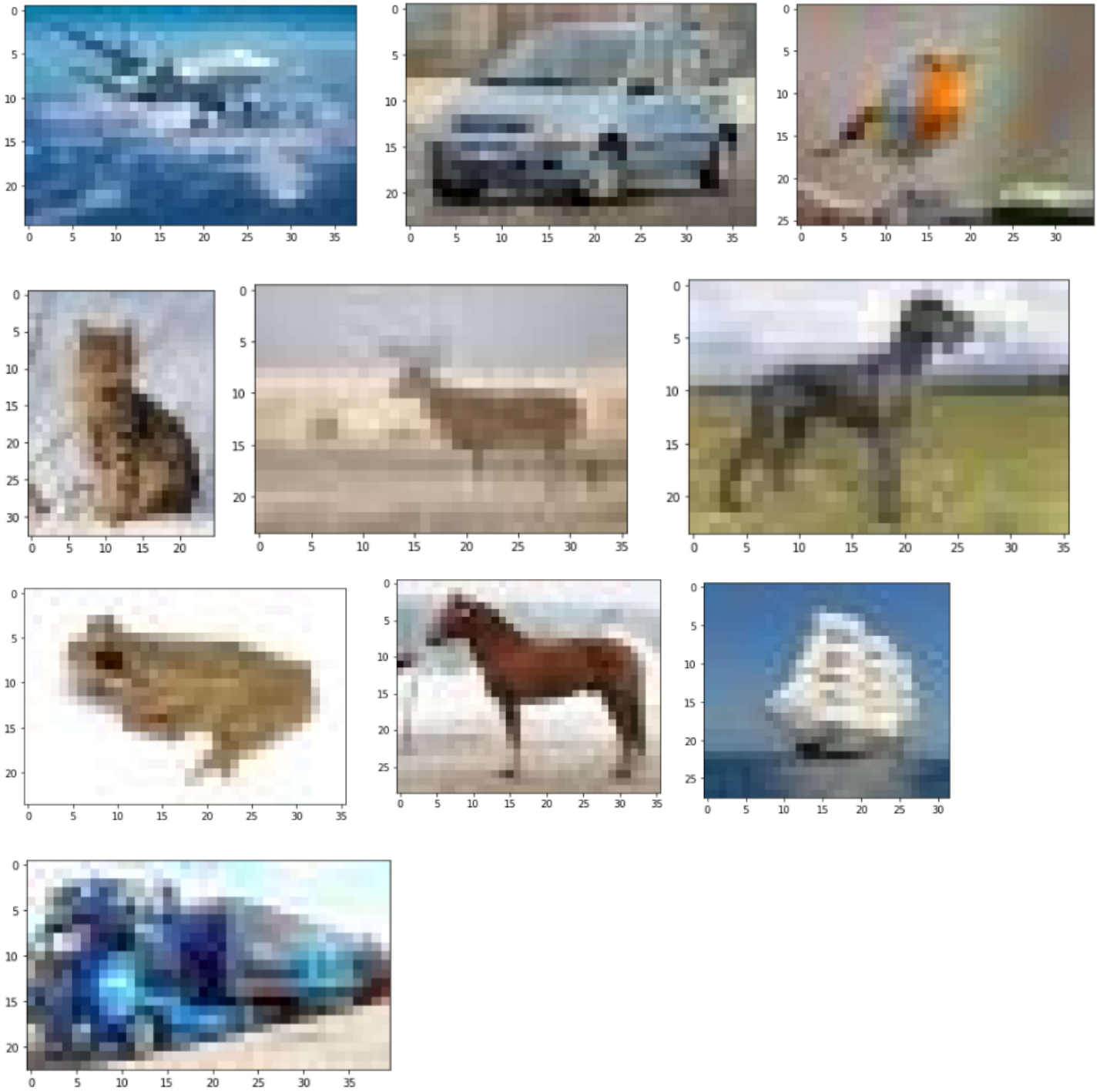
Wykres 7. Trendy optymalizacji modelu z wartwą GAP

3. Wyniki

Wnioski na podstawie wykresów:

- Dodanie kilku bloków konwolucyjnych dało widoczny wzrost poprawy działania modelu, jednak uwidoczniło problem z overfittingiem
- Batch Normalization stabilizuje proces uczenia. Najlepiej widać to po wygładzonych wykresach
- Warstwa Dropout faktycznie pozwala pozbyć się problemów z overfittingiem
- Nie ma dużej różnicy w celności modelu z MaxPoolingiem, a wersji z GAP (pomijając fakt, że GAP daje nam możliwość podania różnych rozmiarów na wejście)

Celem sprawdzenia działania przygotowanego modelu zostały przygotowane następujące zdjęcia:



Rysunek 2. Zdjęcia do werfykacji działania modelu

Sieć rozpoznała zdjęcia na zadowalającym poziomie:

```
Pred:  4 True:  0
Pred:  1 True:  1
Pred:  2 True:  2
Pred:  3 True:  3
Pred:  3 True:  4
Pred:  7 True:  5
Pred:  6 True:  6
Pred:  7 True:  7
Pred:  8 True:  8
Pred:  9 True:  9
```

Wyniki rozpoznawania obrazów za pomocą wytrenowanej sieci (Pred - klasa z największym prawdopodobieństwem)