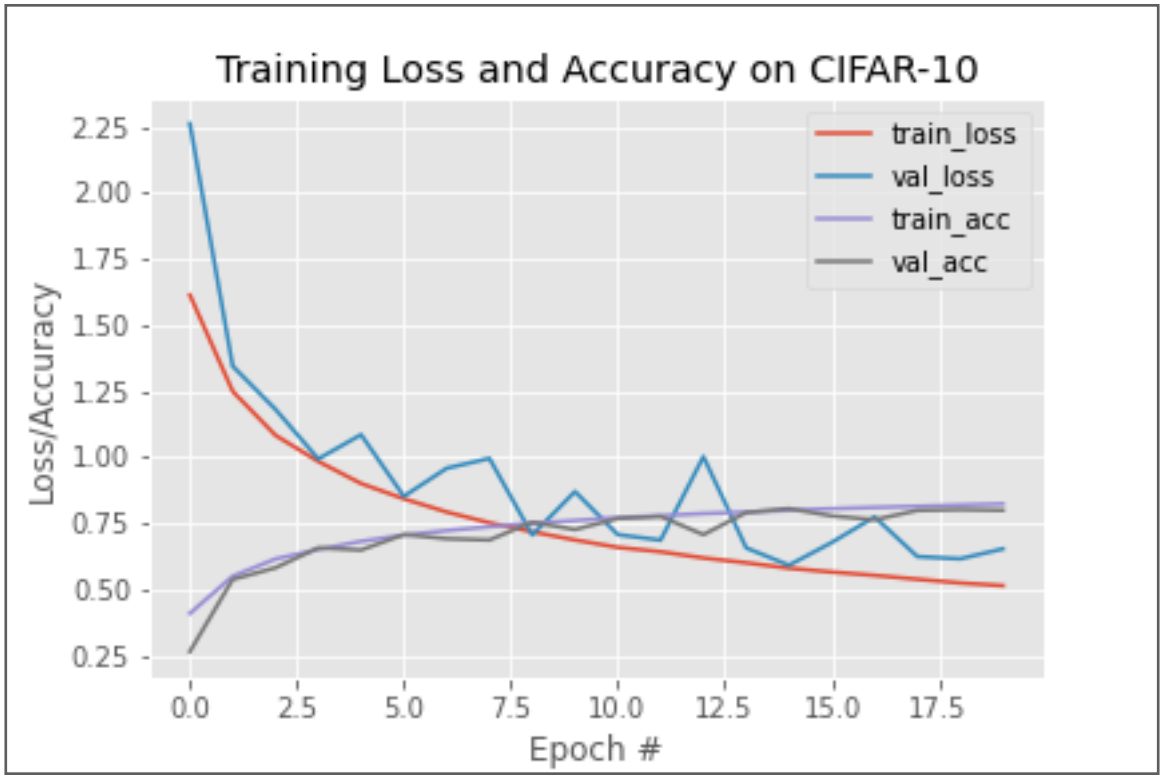
Metody rozpoznawania obrazu
Raport 5
Oscar Teeninga

1. Trening sieci MiniVGGNetModel za pomocą zbioru CIFAR-10

   Zgodnie z poradnikiem stworzyłem sieć, następnie ją przetrenowałem na 20 epokach, w każdej znajdowało się 390 wektorów treningowych. Przetrenowanie każdej zajmowało w okolicy 35s, dzięki obliczeniom wykonywanym w chmurze na GPU na Colab. Wygenerowałem wykres:



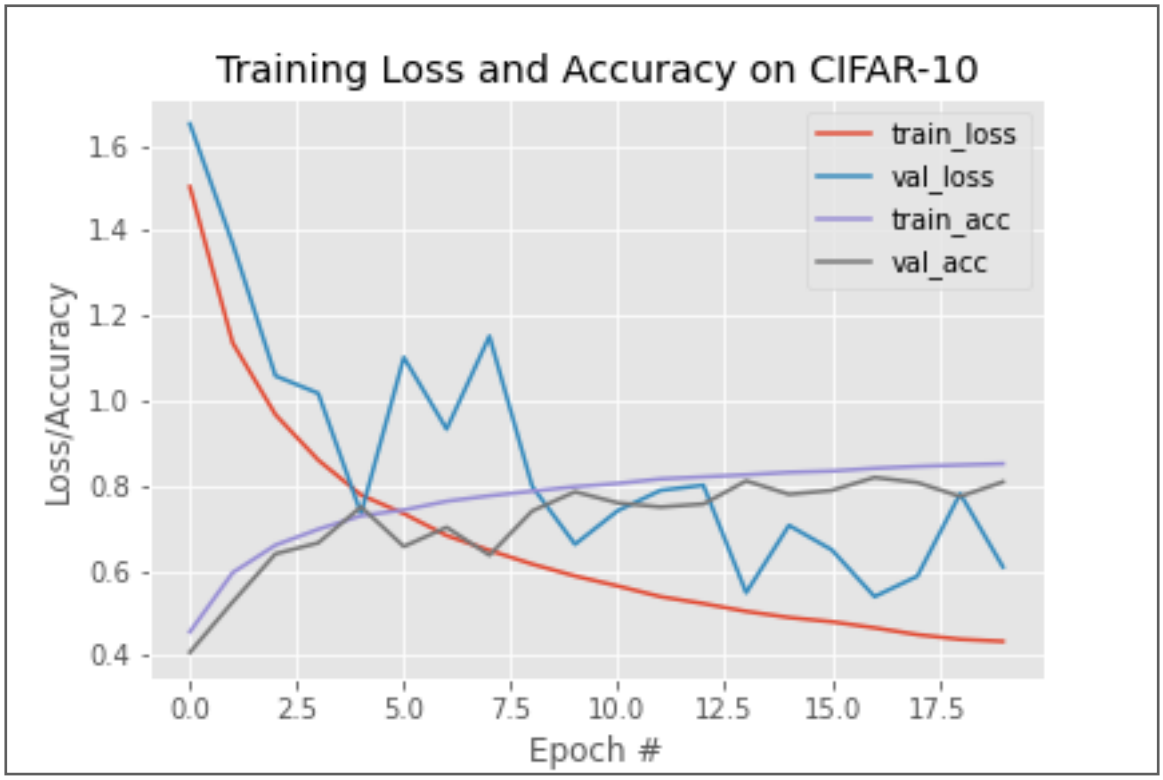
   Udało się osiągnąć zadowalające rezultaty, gdzie accuracy dla zbioru treningowego wyniósł średnio 82%. Widzimy również, że wykres ma kształt eliptyczny co oznacza, że przy większej liczbie epok można by było osiągnąć jeszcze wyższą jakość klasyfikacji i jeszcze nie overfitowaliśmy modelu.

2. Wyrzucenie dropoutu

Aby wykluczyć warstwę dropouty wystarczy w zakomentować:

```
# x = Dropout(0.5)(x).
```

Sprawi to, że wektor wejściowy na etapie analizy nie zostanie przetworzony przez warstwę dropout, a o to nam chodziło. Czas trenowania jednej epoki na granicy błędu pomiarowego, więc wzrost wydajności pomijalny poprzez pominięcie jednej warstwy. Natomiast jeżeli chodzi o wyniki loss/accuracy to prezentują się następująco:



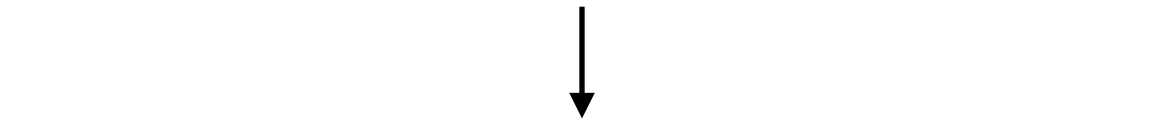
Wyniki są ciekawe. Dropout oznacza, że na pewnym etapie w sieci odrzucamy pewne neurony w celu uniknięcia overfittingu. Nie mieliśmy do czynienia wcześniej z takowym przetrenowaniem, a więc zysk z używania tej warstwy również nie ujrzał światła dziennego, wręcz przeciwnie, usunięcie sprawiło, że udało się osiągnąć wyższy accuracy oscylujący w granicy 83%, natomiast loss dla zbioru treningowego osiągnął wybitne 0.42, co jest znaczną poprawą względem sieci z dropoutem.

W takim wypadku w dalszej analizie będą korzystał z tej wersji sieci bez dropoutu.

3. Separable convolutions

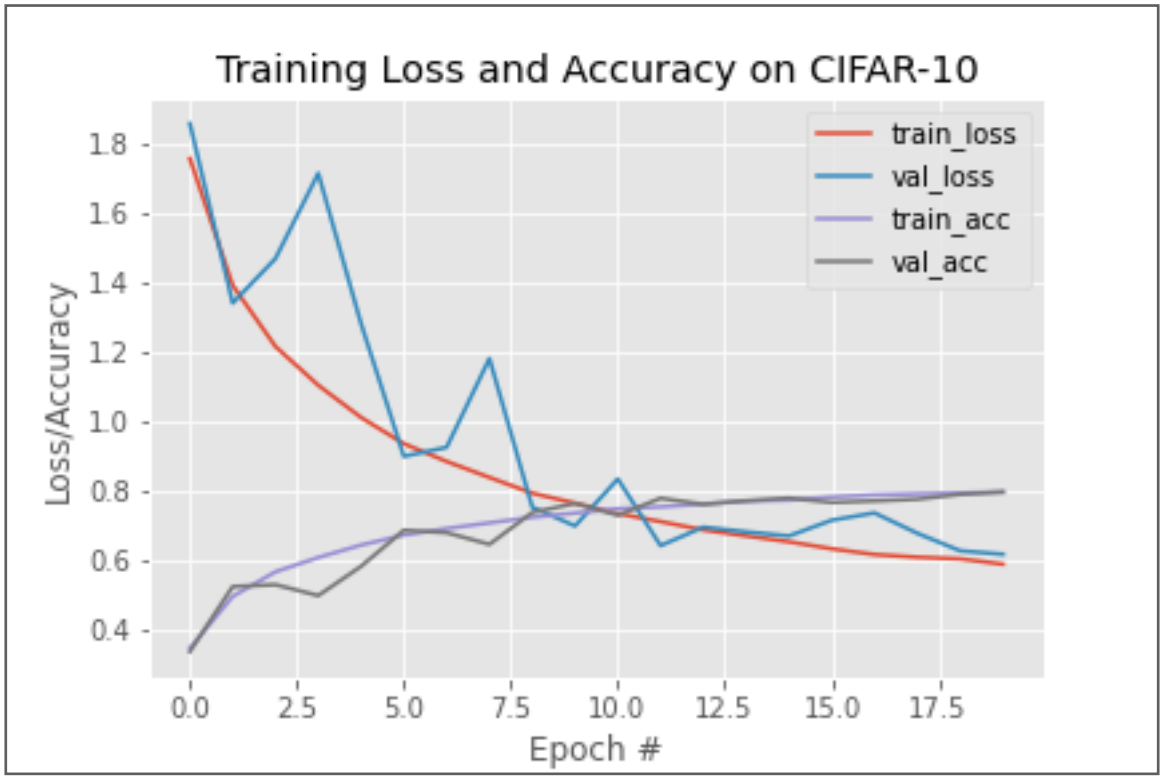
Następnie przeszedłem do rozdzielenia konwolucji. W celu zrozumienia zagadnienia (musiało mi umknąć na labach) posiłkowałem się poradnikiem (link). Zgodnie z tym co jest tam napisane, zamiast przeprowadzać jedną konwolucję, rozbijamy ją na dwie:

```
x = conv_module(x, numK3x3, 3, 3, (1, 1), chanDim)
```



```
x = conv_module(x, numK3x3, 3, 1, (1, 1), chanDim)
x = conv_module(x, numK3x3, 1, 3, (1, 1), chanDim)
```

Po raz kolejny przeprowadziłem trening i wyniki klasyfikacji prezentują się następująco:



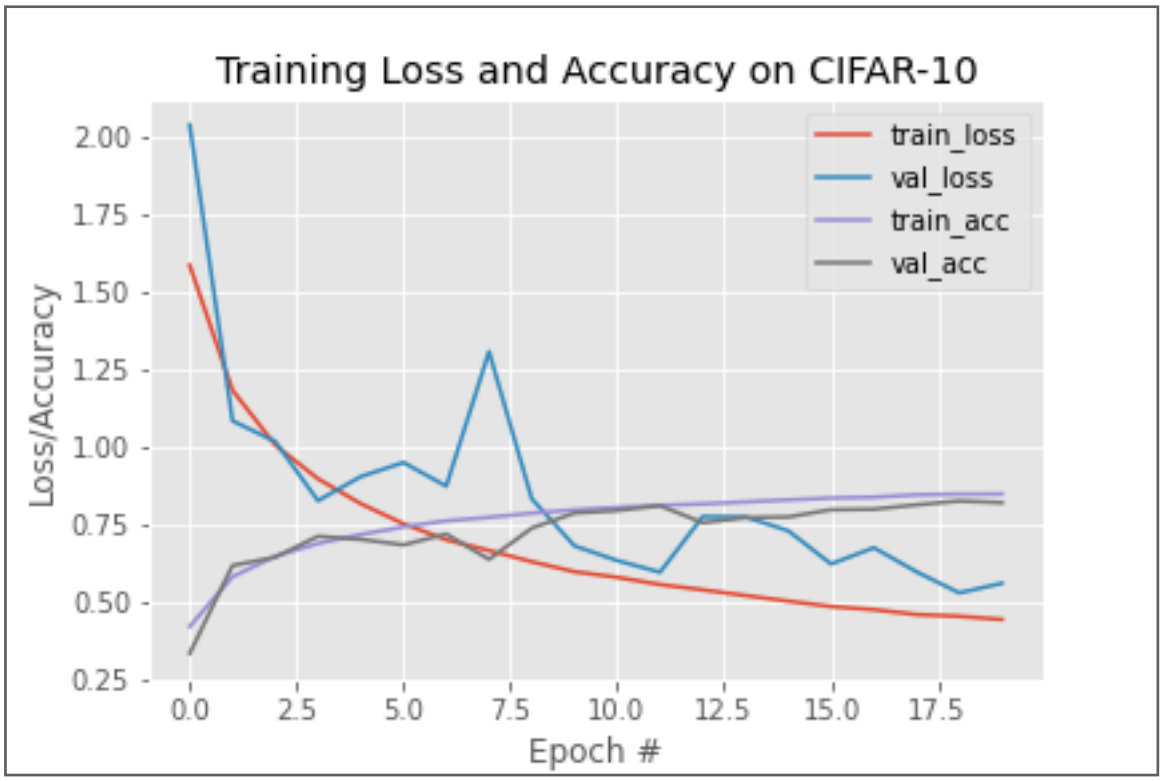
Czas trenowania jednej epoki zmniejszył się do 30 sekund, ale jak widzimy jest to okupione gorszą jakością, odpowiednio 80% dla accuracy i 0.6 loss. Poprawę odrzucam.

4. Zastąpienie konwolucji 1x1/3x3 przez 5x5

   Zgodnie z instrukcją, liczba konwolucji pozostaje stała, jedna piąta bazowych konwolucji zostaje zastępiona 5x5:

```
conv_1x1 = conv_module(x, int(numK1x1*4/5), 1, 1, (1, 1), chanDim)
conv_3x3 = conv_module(x, int(numK3x3*4/5), 3, 3, (1, 1), chanDim)
conv_5x5 = conv_module(x, int(numK3x3/5+numK1x1/5), 5, 5, (1, 1), chanDim)
x = concatenate([conv_1x1, conv_3x3, conv_5x5], axis=chanDim)
```

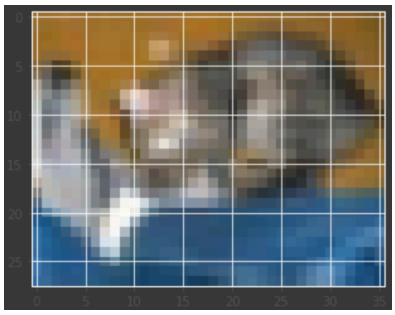
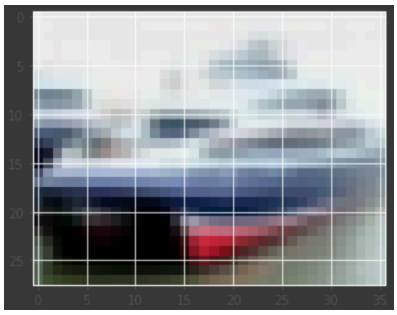
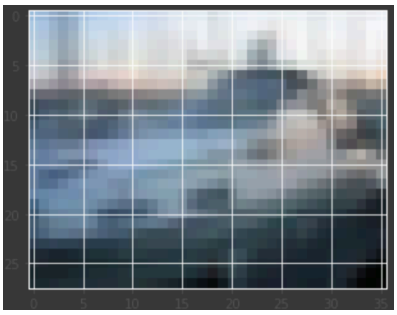
   A następnie przetrenowałem model. Trening jednej epoki trwał około 54s, a więc prawie dwa razy dłużej, co było raczej oczekiwane. Wygenerowałem wykres:



   Zmiany sprawiły, że średnia dokładność wynosiła najwyższe do rej pory 84%, natomiast okupione jest to znacznym wzrostem czasu uczenia, więc do następnych analiz odrzucam poprawkę.

5. Dowolny obraz wejściowy

Skorzystałem z trzech pierwszych obrazków zbioru testowego:
Rozmiar każdego to odpowiednio 36x28 - jest to kotek i dwa statki.

Do zmiany wystarczyło zaimportować GlobalAveragePooling2D oraz podmienić za zwykły AveragePooling2D oraz usunąć argument (7, 7).
Skorzystałem z cv2.resize(image, dim) do zmiany rozmiaru obrazów.
Zgodnie z oczekiwaniami, udaje się poprawnie zakwalifikować obrazy, najpierw wywołanie model.prediction(images):

```
array([[2.2213054e-03, 4.9848517e-04, 5.4269809e-02, 7.6645792e-01,
        4.1359249e-03, 8.4563777e-02, 8.5475184e-02, 2.8893116e-04,
        2.0183269e-03, 7.0387461e-05],
       [2.2637725e-04, 4.8231077e-03, 5.3465925e-07, 7.8117046e-07,
        7.5678048e-07, 1.0489500e-07, 4.1742163e-07, 4.2542769e-07,
        9.9493068e-01, 1.6869542e-05],
       [1.3003650e-04, 1.5998476e-03, 4.7704198e-06, 5.6204240e-06,
        1.6715107e-05, 3.3022798e-07, 8.1673528e-07, 5.4216730e-07,
        9.9821281e-01, 2.8588582e-05]], dtype=float32)
```

Co daje odpowiednio:

```
array([3, 8, 8])
```

Co oznacza [kot, statek, statek], a więc w pełni poprawnie.