WDUM Projekt sprawozdanie

Kamil Kulesza

December 27, 2022

1 Klasyfikacja obrazów zwierząt

Cel klasyfikacja obrazów zwierząt do jednej z dziesięcu klas:

$$[butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, squirrel]$$

za pomocą **Convolutional Neural Networks (CNN)**. Dla każdej klasy jest około 2 tysiące obrazów.

Źródło danych: https://www.kaggle.com/datasets/alessiocorrado99/animals10

[7]:
```
import torch
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torchvision.models import resnet18, alexnet
from torch import nn
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
import utils
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

torch.manual_seed(423)
device = torch.device(0) if torch.cuda.is_available() else torch.device('cpu')
TRAIN_MEAN = [0.5036, 0.4719, 0.3897]
TRAIN_STD = [0.2623, 0.2577, 0.2671]
classes = ['butterfly','cat', 'chicken', 'cow', 'dog', 'elephant', 'horse',␣
 ↪'sheep', 'spider', 'squirrel']
```

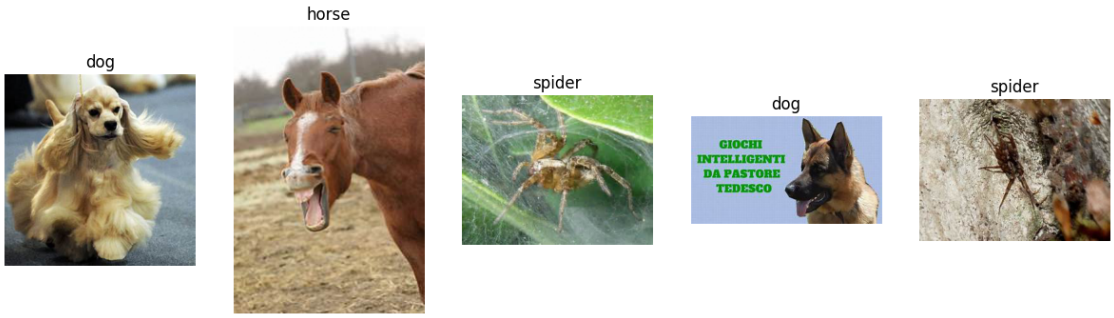
1.1 Problemy związane z przyjęciem obrazów jako wejscie dla modeli

1.1.1 Przykładowe obrazy w zbiorze

[8]:
```
transform = transforms.Compose([
    transforms.ToTensor(),
])
```

```
train_data = ImageFolder(root='../dataset/train/', transform=transform)
batch_size = 1
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
```

[9]:
```
fig, ax = plt.subplots(1, 5, figsize=(15,15))
for i in range(5):
    data, labels = next(iter(train_loader))
    img = data[0].permute(1,2,0)
    ax[i].imshow(img)
    ax[i].axis('off')
    ax[i].set_title(classes[labels[0].item()])
```



Podobnie jak w przypadku danych tabelarycznych oczekujemy pewnego określonego wymiaru danych wejsciowych. W tym przypadku każdy obraz możemy traktowac jako tabele o wymiarach $H \times W \times C$ - wysokość, szerokość i kanały koloru (RGB), a informacją jest wartość pixela $[0-255]$ (lub $[0-1]$ po znormalizowaniu).

[10]:
```
data[0].shape, data[0,:, :5, :5],
```

[10]:
```
(torch.Size([3, 223, 300]),
 tensor([[[0.7294, 0.7137, 0.7843, 0.7529, 0.8275],
          [0.7490, 0.7412, 0.7882, 0.7333, 0.7882],
          [0.7216, 0.7373, 0.7765, 0.7333, 0.7882],
          [0.7020, 0.7373, 0.7608, 0.7412, 0.7922],
          [0.7412, 0.7647, 0.7569, 0.7608, 0.7725]],

         [[0.7020, 0.6863, 0.7569, 0.7255, 0.7922],
          [0.7216, 0.7137, 0.7608, 0.7059, 0.7529],
          [0.6941, 0.7098, 0.7490, 0.7059, 0.7529],
          [0.6745, 0.7098, 0.7333, 0.7137, 0.7569],
          [0.7137, 0.7373, 0.7294, 0.7333, 0.7373]],

         [[0.6392, 0.6235, 0.6941, 0.6627, 0.7255],
          [0.6588, 0.6510, 0.6980, 0.6431, 0.6863],
          [0.6314, 0.6471, 0.6863, 0.6431, 0.6863],
```

```
        [0.6118, 0.6471, 0.6706, 0.6510, 0.6902],
        [0.6510, 0.6745, 0.6667, 0.6706, 0.6706]]]))
```

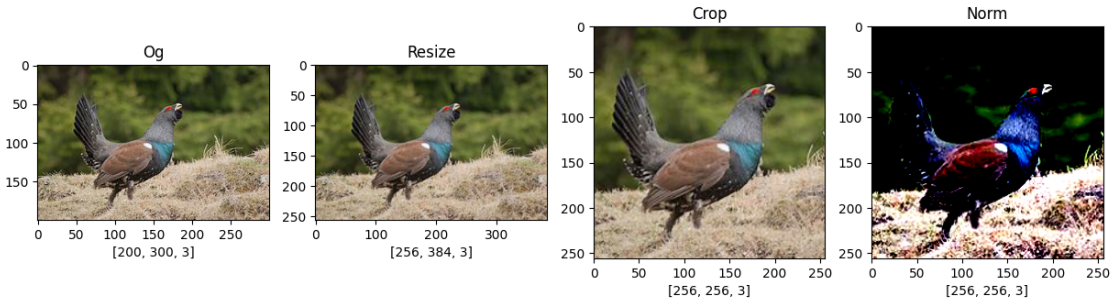
1.1.2 Preprocessing obrazów

Poniewaz obrazy mają różne wymiary przyjęto podejście zwiększenia najmniejszego z wymiarów do $[256 \times W]$ lub $[H \times 256]$, a drugi wymiar zwiększany jest zgodnie z zachowaniem aspect ratio oryginalnego obrazu co pozwala na uniknięcie zniekształceń kształtów (np. spłaszczenie głowy). Natomiast chcemy mięć obrazy o stałym wymiarze np. $[256 \times 256]$ więc robimy losowe wycięcie takiego obszaru z powiększonego obrazu. Dodatkowa transformacja dla polepszenia procesu uczenia sieci to standaryzacja wartosci pixeli.

[11]:
```
data, labels = next(iter(train_loader))
fig, ax = plt.subplots(1, 4, figsize=(15,15))
# oryginalny obraz
og_img = data[0].permute(1,2,0)
# resize obrazu
resize = transforms.Resize(256)(data)
resized_img = resize[0].permute(1,2,0)
# crop obrazu
crop= transforms.RandomCrop(size=(256,256))(resize)
cropped_img = crop[0].permute(1,2,0)
# normalizacja
norm = transforms.Normalize(TRAIN_MEAN, TRAIN_STD)(crop)
norm_img = norm[0].permute(1,2,0)

names = ['Og', 'Resize', 'Crop', 'Norm']
for i, (im, name) in enumerate(zip([og_img, resized_img, cropped_img,␣
 ↪norm_img], names)):
    ax[i].imshow(im)
    ax[i].set_title(name)
    ax[i].set_xlabel(list(im.shape))
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



1.1.3 Filtry liniowe i operacja konwolucji

Filtorwanie obrazu w celu pozyskania pewnych cech z obrazów. W najprostszy sposób możemy po prostu spłaszczyć macierz obrazu w jednowymiarowy wektor, ale tracimy informacje o lokalnych cechach wystepujących w obrazie.

Sposobem na przetworzenie obrazów z wykorzystaniem lokalnych filtrów jest operacja **konwolucji** obrazu z jądrem (kernel), przykładowo blurowanie obrazu.

[12]:
```
fig, ax = plt.subplots(1, 5, figsize=(15,15))

og_img = Image.fromarray((cropped_img.numpy()*255).astype('uint8'))
ax[0].imshow(cropped_img)
ax[0].set_title(f'Og')
ax[0].axis('off')

for i, ker_size in enumerate([3,5,7,9]):
    im = og_img.filter(ImageFilter.BoxBlur(ker_size))
    im = np.array(im)
    ax[i+1].imshow(im)
    ax[i+1].axis('off')
    ax[i+1].set_title(f'Kernel size: {ker_size}')
```

Przykładowe kernele 3×3 i 5×5 uśredniające wartości pixeli na lokalnym obszarze obrazu. Wartość sumuje się do jedynki, by nie wzmacniać/usłabiać wartości na danym obszarze.

[13]:
```
ker_3, ker_5 = np.ones(shape=(3,3))/9, np.ones(shape=(5,5))/25
ker_3, ker_5
```

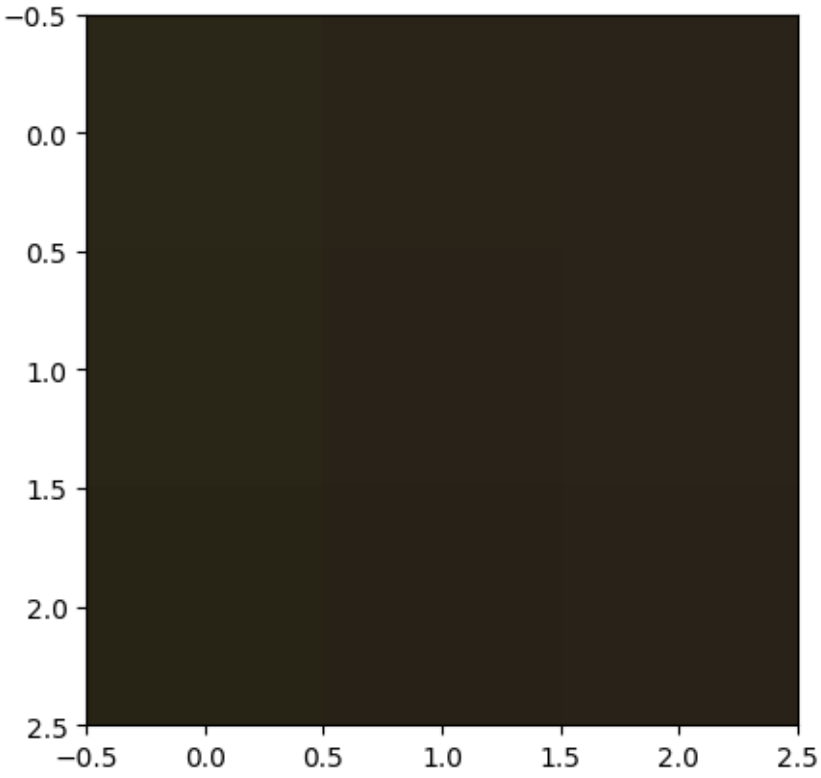
[13]:
```
(array([[0.11111111, 0.11111111, 0.11111111],
        [0.11111111, 0.11111111, 0.11111111],
        [0.11111111, 0.11111111, 0.11111111]]),
 array([[0.04, 0.04, 0.04, 0.04, 0.04],
        [0.04, 0.04, 0.04, 0.04, 0.04],
        [0.04, 0.04, 0.04, 0.04, 0.04],
        [0.04, 0.04, 0.04, 0.04, 0.04],
        [0.04, 0.04, 0.04, 0.04, 0.04]]))
```

[14]:
```
img = (cropped_img.numpy()*255).astype('uint8')[:3, :3, :]
print(img)
plt.imshow(img)
plt.show()
```

```
[[[42 38 24]
  [42 35 24]
  [42 35 25]]

 [[41 37 23]
  [41 34 24]
  [42 35 25]]

 [[39 36 21]
  [40 33 23]
  [41 34 24]]]
```

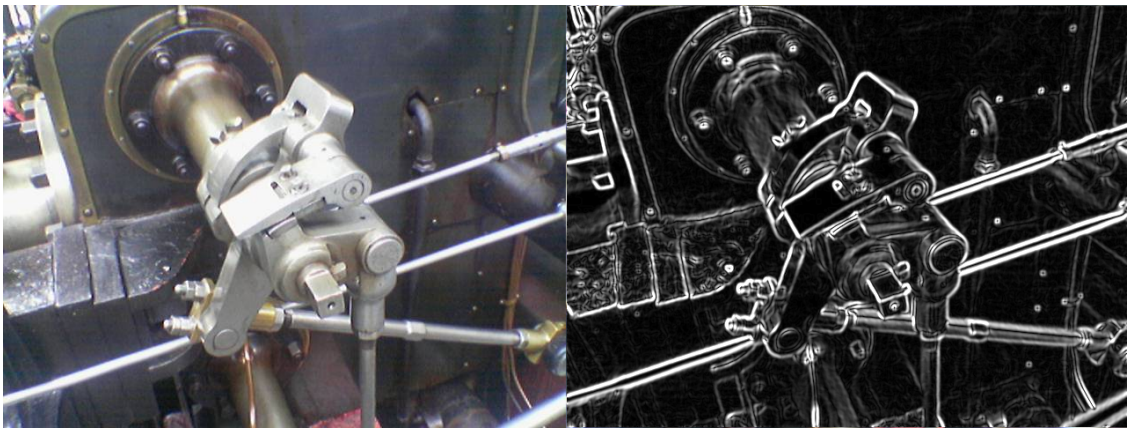


Wynikające wartości pixela na kanałach RGB z konwolucji, dla każdego z kanałów. Jest to wartość wyłącznie dla pixela H,W = (1,1) na podstawie jego otoczenia. Taka operację nalezy przeprowadzić na otoczeniach wszystkich pixeli, aby uzyskać zblurowany obraz.

[15]:
```
np.einsum('jk, ijk -> k', ker_3, img).astype('uint8')
```

[15]:
```
array([41, 35, 23], dtype=uint8)
```

Specjalistyczne ręcznie dobrane filtry do wykrywania wybranych cech jak np. krawędzie obiektów w obrazie (Sobel Filter), są one inspiracją dla sieci typu CNN, aby automatycznie uczyły się takich filtrów.



Źródło obrazów: https://en.wikipedia.org/wiki/Sobel_operator

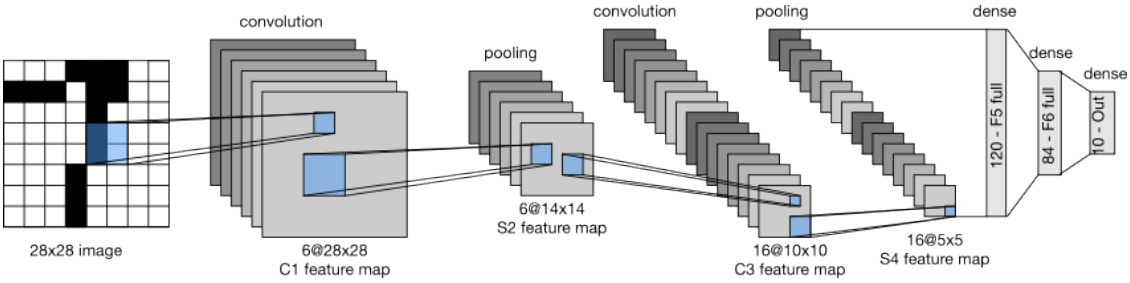
[16]:
```
np.array([[1,0,-1], [+2, 0, -2], [1,0,-1]]), np.array([[1,2,1], [0, 0, 0],␣
↪[-1,-2,-1]])
```

[16]:
```
(array([[ 1,  0, -1],
        [ 2,  0, -2],
        [ 1,  0, -1]]),
 array([[ 1,  2,  1],
        [ 0,  0,  0],
        [-1, -2, -1]]))
```

1.2 Sieci CNN

Architektura prostej sieci CNN (LeNet):

1. Ekstrakcja cech - warstwa konwolucyjna, warstwy z filtrami o różnej wielkości. Celem jest, aby sieć by sama nauczyla się potrzebnych jej filtrów do rozróżniania cech danych klas.
2. Klasyfikator - koncepcyjnie mamy np. 200 filtrów w ostatniej warstwie konwolucyjnej, "spłaszczamy" informację do 1D tensora, np. binarna informacja 0-1 dla filtru o kształcie kocich uszu.
3. Dalej standardowa sieć neuronowa z celem klasyfikacji obiektu do jednej z 10 klas.



Źródło obrazu: http://d2l.ai/chapter_convolutional-neural-networks/lenet.html

4. Wyjście sieci - funkcja **softmax** by informacje liczbową wyjść przedstawić w formie prawdopodobieństwa przynależności do klasy. Taki wektor po wyjściu z funkcji softmax sumuje się do 1, a przynależność do klasy określamy po indeksie maksymalnej wartości.

$$softmax(y) = \left[\frac{e^{y_1}}{\sum_j^C e^{y_j}}, \frac{e^{y_2}}{\sum_j^C e^{y_j}}, \dots, \frac{e^{y_C}}{\sum_j^C e^{y_j}} \right]$$

5. Funkcja celu - **Cross Entropy Loss**

$$CE(P^*|P) = -\sum_j^C P^*(j) \log P(j)$$

- P^* - wektor prawdziwych wartości
- P - wektor predykcji

Co uproszcza się do $-\log P(c)$, gdyż wektor P^* przyjmuje postać $[0, \dots, 1, \dots, 0]$ więc pozostałe elementy sumy się zerują.

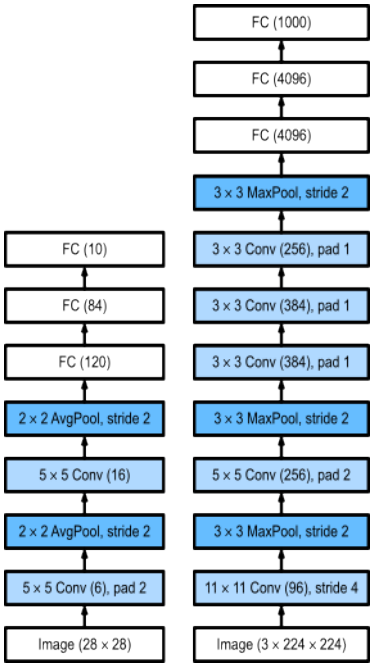
1.2.1 Uczenie sieci

- Algorytm optymalizacja wag za pomocą **Mini Batch Stochastic Gradient Descent** z regularyzacją wag
- Batch Size - 32 obrazy per batch, okolo 560 batchy, kwestia pomieszczenia wag parametrów sieci i obrazów w pamięci GPU, by przyśpieszyć obliczenia
- Nauczono trzy architektury sieci: **LeNet**, **AlexNet**, **ResNet**
- Dwa podejścia uczenia wag: od zera z losowymi wagami oraz transfer learning

Transfer learning - sieć z wagami nauczonymi na zbiorze ImageNet (1 mln obrazów, 1000 klas), polega to na dostosowaniu wyjścia sieci do problemu (1000 -> 10) i "douczenia" wcześniejszych warstw z mniejszym krokiemm niż warstwa wyjściowa. Takie podejście jest bardzo efektywne, gdyż te sieci zostały nauczone do osiągnięcia wysokiej skuteczności wcześniej i upublicznione, a wyuczone filtry dostoswują się do nowego problemu.

1.2.2 LeNet i AlexNet

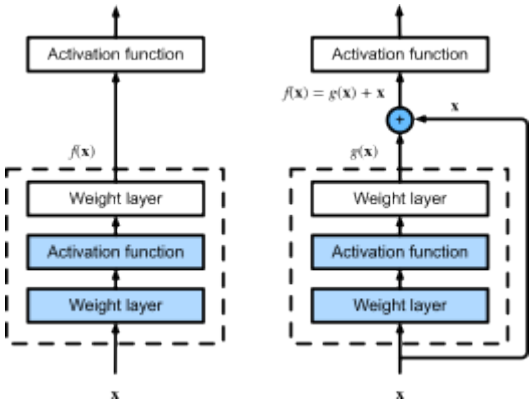
Architektura sieci:



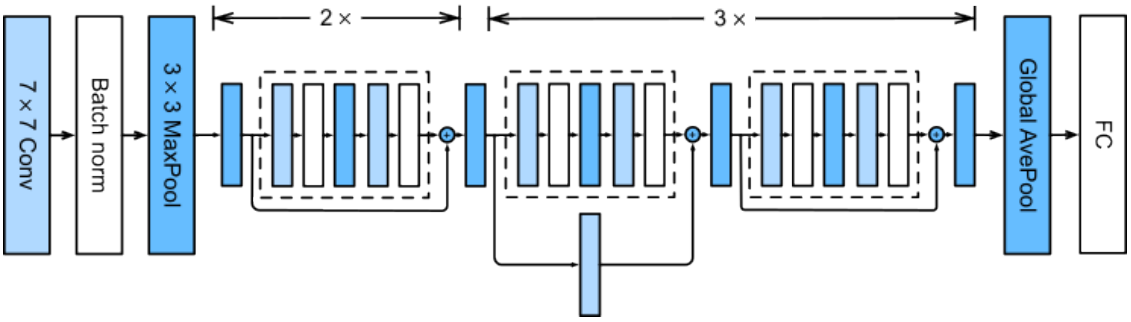
Źródło obrazu: http://d2l.ai/chapter_convolutional-modern/alexnet.html#alexnet

1.2.3 ResNet

Residual block - uproszcza uczenie sieci zakreskowany obszar nie musi sie uczyc całej funkcji $f(x)$, a uczy się uproszczonej wersji a reszta jest dociągana do wyjścia. Upraszcza to bo przed funkcja aktywacją komibnacja wyjścia jest sumą kombinacji liniowych.



Architektura sieci:



Źródło obrazów http://d2l.ai/chapter_convolutional-modern/resnet.html

1.3 Wyniki

[30]:
```
df = pd.read_csv('../models/results.csv')
session_ordered = [
    '20221209_230012', '20221209_213015', '20221209_215731', '20221209_211433',␣
 ↪'20221209_195344', '20221222_085837'
]
session_map = {
    '20221209_230012': 'LeNet - Random Weights',
    '20221209_213015': 'AlexNet - Random Weights',
    '20221209_215731': 'AlexNet - ImgNet Weights',
    '20221209_211433': 'ResNet - Random Weights',
    '20221209_195344': 'ResNet - ImgNet Weights',
    '20221222_085837': 'ResNet - Long training '
}
```

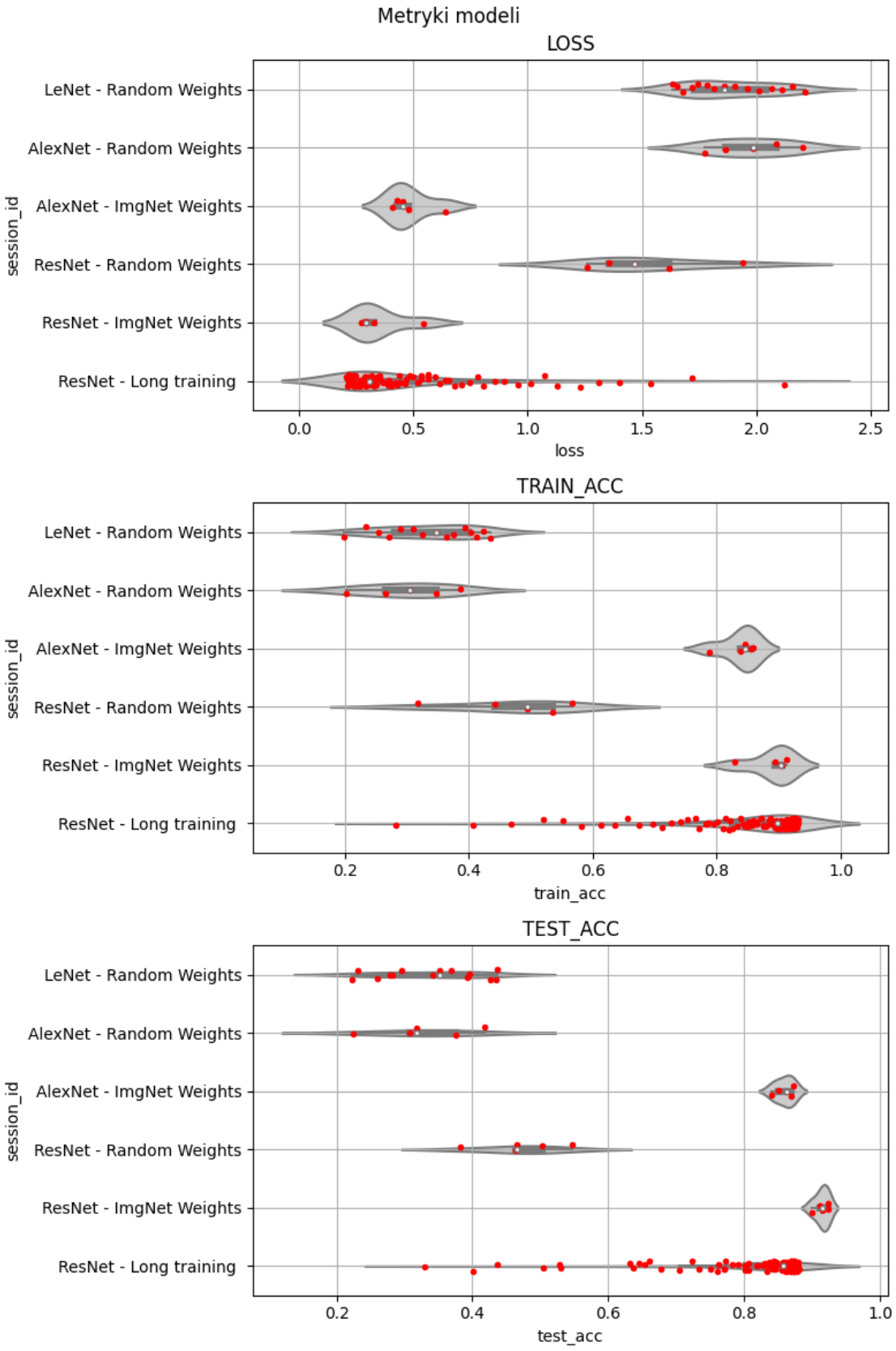
1.3.1 Porównanie metryk miedzy modelami

Przyjęte metryki - wartość funkcji celu na zbiorzu uczącym - dokładność predykcji na zbiorze uczącym - dokładność predykcji na zbiorze testowym

[31]:
```
x = df.copy()
# sortowanie po session id
x.session_id = x.session_id.astype('category').cat.
 ↪set_categories(session_ordered)
x = x.sort_values(['session_id'])
# przypisanie nazw sesjom
x.session_id = x.session_id.map(session_map)

fig, ax = plt.subplots(3, 1, figsize=(8, 12), tight_layout=True)
fig.suptitle('Metryki modeli')
for i, metric in enumerate(['loss', 'train_acc', 'test_acc']):
    sns.stripplot(data=x, x=metric, y="session_id", size=4, color='red',␣
 ↪ax=ax[i])
    sns.violinplot(data=x, x=metric, y="session_id", color=".8", ax=ax[i])
```

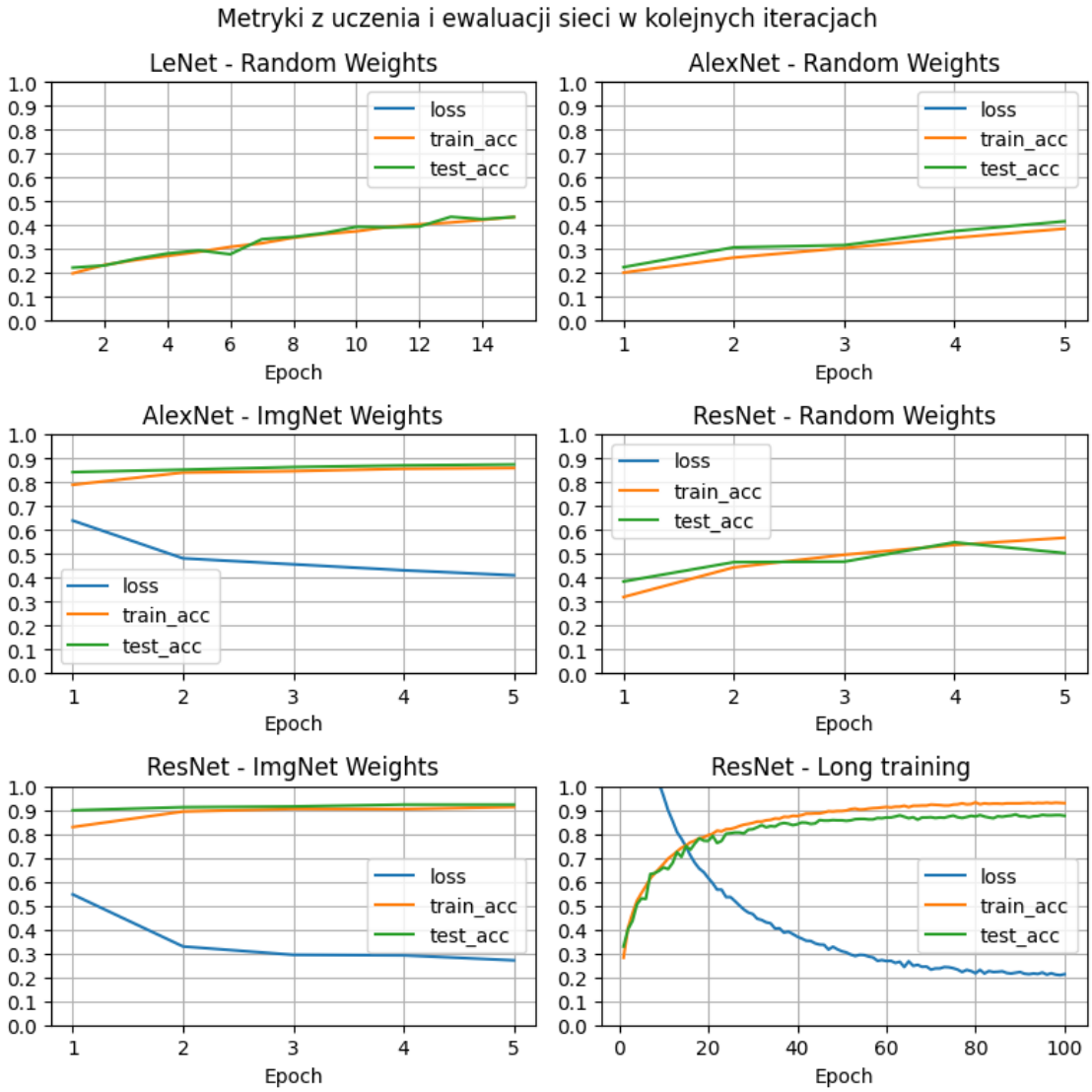
```
    ax[i].set_title(metric.upper())
    ax[i].grid(True)
```


Metryki modeli
LOSS
LeNet - Random Weights
AlexNet - Random Weights
AlexNet - ImgNet Weights
ResNet - Random Weights
ResNet - ImgNet Weights
ResNet - Long training
session_id
0.0
0.5
1.0
1.5
2.0
2.5
loss
TRAIN_ACC
LeNet - Random Weights
AlexNet - Random Weights
AlexNet - ImgNet Weights
ResNet - Random Weights
ResNet - ImgNet Weights
ResNet - Long training
session_id
0.2
0.4
0.6
0.8
1.0
train_acc
TEST_ACC
LeNet - Random Weights
AlexNet - Random Weights
AlexNet - ImgNet Weights
ResNet - Random Weights
ResNet - ImgNet Weights
ResNet - Long training
session_id
0.2
0.4
0.6
0.8
1.0
test_acc

1.3.2 Zmiana w wartościach metryk w kolejny epokach

[33]:
```
fig, ax = plt.subplots(3, 2, figsize=(8,8))

for i in range(3):
    for j in range(2):
        session = session_ordered[2*i+j]
        ax[i, j].plot(df[df.session_id == session].epoch, df[df.session_id ==␣
 ↪session].loss)
        ax[i, j].plot(df[df.session_id == session].epoch, df[df.session_id ==␣
 ↪session].train_acc)
        ax[i, j].plot(df[df.session_id == session].epoch, df[df.session_id ==␣
 ↪session].test_acc)
        ax[i, j].set_ylim([0, 1])
        ax[i, j].set_yticks(np.linspace(0,1,11))
        ax[i, j].set_title(session_map[session])
        ax[i, j].legend(['loss', 'train_acc', 'test_acc'])
        ax[i, j].set_xlabel('Epoch')
        ax[i, j].grid(True)
fig.suptitle("Metryki z uczenia i ewaluacji sieci w kolejnych iteracjach")
fig.tight_layout()
```



1.3.3 Porównanie jakości klasyfikacji ResNet - ImageNet weights vs ResNet-Long training

[21]:
```
res_net_model = resnet18()
res_net_model.fc = nn.Linear(res_net_model.fc.in_features, len(classes))

def model_train_evaluation(model_params_file, model):
    # zaladowanie modelu i wczytanie na gpu
    model.load_state_dict(torch.load(model_params_file))
    model.to(device)

    # zdefiniowanie operacji na kazdym obrazie w zbiorze
    transform = transforms.Compose([
```

```
        transforms.RandomResizedCrop(256),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(TRAIN_MEAN, TRAIN_STD)
    ])
    # loader danych z batchami
    test_data = ImageFolder(root='../dataset/test/', transform=transform)
    batch_size = 32
    test_loader = DataLoader(test_data, batch_size=batch_size)

    ## PREDYKCJA OGÓŁEM BEZ PODZIAŁU NA KLASY
    correct = 0
    total = 0
    ## PREDYKCJA Z PODZIAŁEM NA KLASY
    correct_pred = {classname: 0 for classname in classes}
    total_pred = {classname: 0 for classname in classes}
    ## WSZYSTKIE PREDYKCJE I LABELE DO CONFUSION MATRIX
    all_predictions = torch.tensor([], device=device)
    all_labels = torch.tensor([], device=device)

    with torch.no_grad():
        model.eval()
        for images, labels in test_loader:
            # PREDYKCJA
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            predicted = torch.argmax(outputs, 1)
            # SPRAWDZENIE ILE POPRAWNIE
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            # SPRAWDZANIE DLA POSZCZEGOLNYCH KLAS
            for label, prediction in zip(labels, predicted):
                if label == prediction:
                    correct_pred[classes[label]] += 1
                total_pred[classes[label]] += 1
            # SKLEJANIE WSZYSTKICH PREDYKCJI I LABELI
            all_predictions = torch.concat([all_predictions, predicted], dim=0)
            all_labels = torch.concat([all_labels, labels], dim=0)

    # przetowarzenie wynikow
    # accuracy calosciowe
    col_typ = ['All classes']
    col_accuracy = [round(100*correct/total,3)]
    # accuracy z podzialem na klasy
    for classname, correct_count in correct_pred.items():
        accuracy = round(100*correct_count/total_pred[classname], 3)
        col_typ.append(classname)
```

```
        col_accuracy.append(accuracy)
    # confusion matrix
    cm = confusion_matrix(all_labels.cpu().numpy(),all_predictions.cpu().
↪numpy())
    return pd.DataFrame({'Class': col_typ, 'Accuracy': col_accuracy}).
↪set_index('Class') ,cm
```

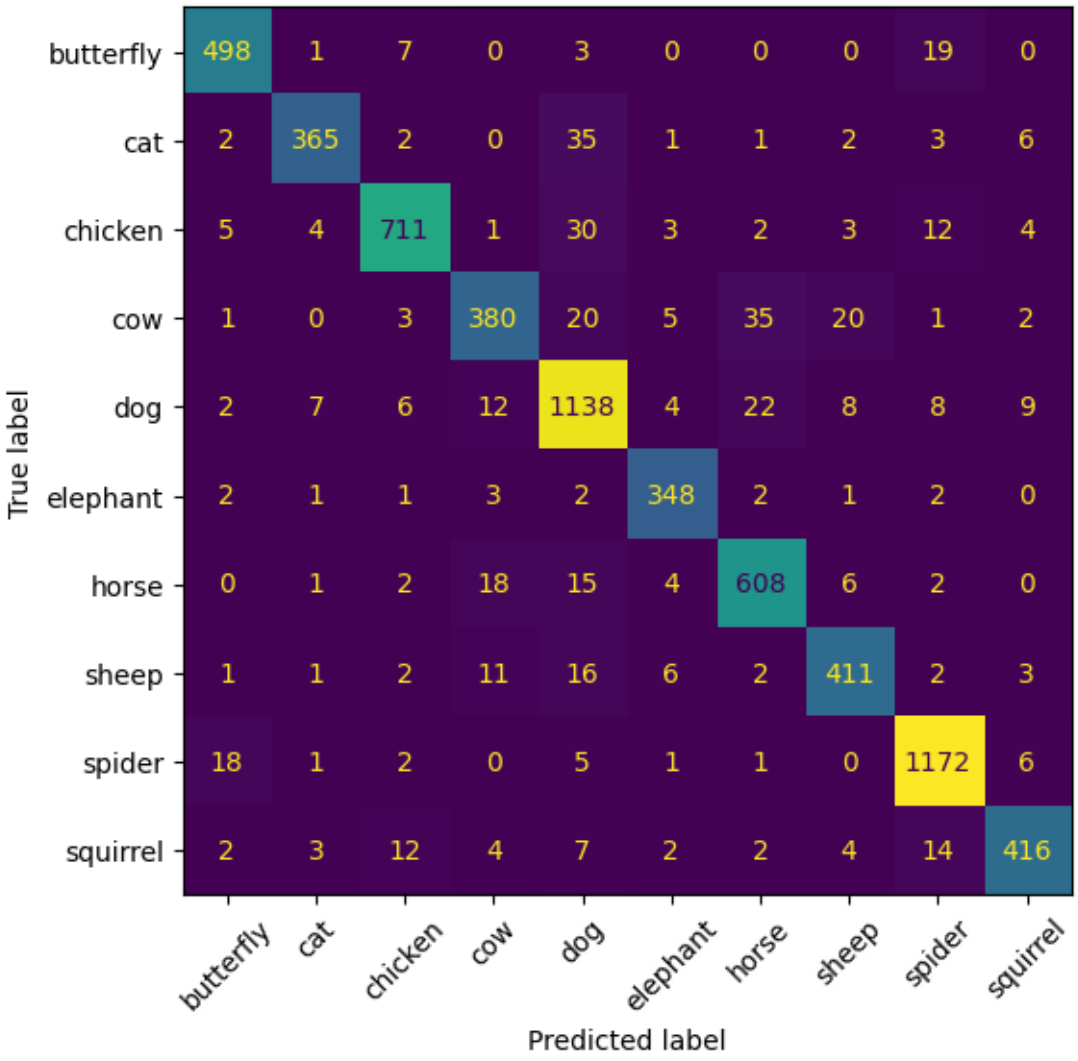
ResNet - ImageNet weights

[27]:
```
df, cm = model_train_evaluation('../model_params/train20221209_195344_5',␣
↪res_net_model)
df
```

[27]:

Class	Accuracy
All classes	92.349
butterfly	94.318
cat	87.530
chicken	91.742
cow	81.370
dog	93.586
elephant	96.133
horse	92.683
sheep	90.330
spider	97.181
squirrel	89.270

[28]:
```
fig, ax = plt.subplots(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(ax=ax, colorbar=False, xticks_rotation='45')
plt.show()
```



ResNet - Long training

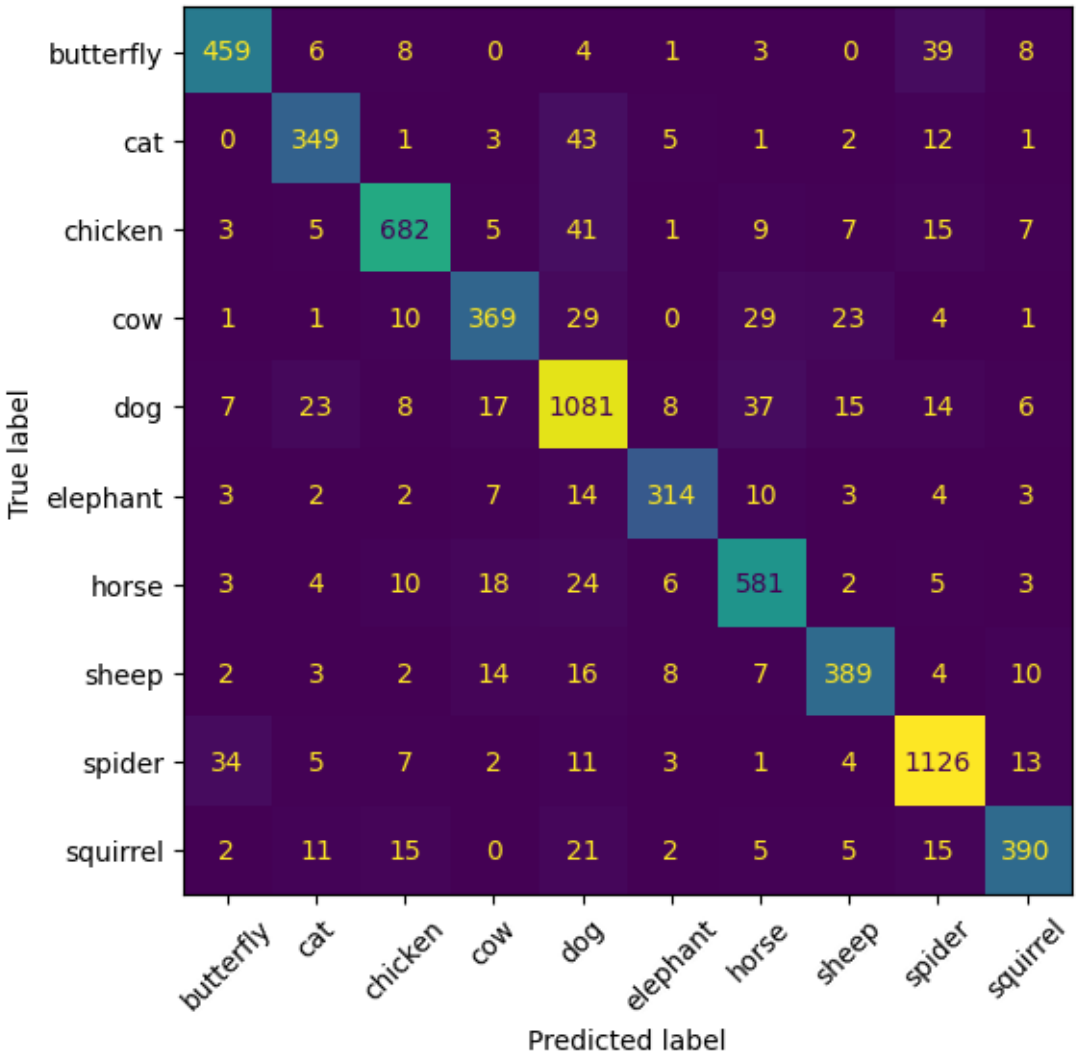
[22]:
```
df, cm = model_train_evaluation('../model_params/20221222_085837',␣
↪res_net_model)
df
```

[22]:

Class	Accuracy
All classes	87.660
butterfly	86.932
cat	83.693
chicken	88.000
cow	79.015
dog	88.898

```
elephant        86.740
horse           88.567
sheep           85.495
spider          93.367
squirrel        83.691
```

[23]:
```
fig, ax = plt.subplots(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(ax=ax, colorbar=False, xticks_rotation='45')
plt.show()
```



1.4 Wnioski:

- Wytrenowanie własnego modelu do zadań klasyfikacji obrazu jest czasochłonne. Trenowanie sieci przez 100 epok zajeło prawie 5 godzin, a ostateczny model na zbiorze testowym osiąga o 7 punktów procentowych niższa dokładność niż dotrenowany model z wagami zbioru ImageNet.
- W obu przypadkach błąd nieprawidłowej klasyfikacji wystepuje rzadko.
- Inni badacze na podobnych architekturach osiągali podobne wyniki (co jest kwestią wygody związane z zastosowaniem metody transfer lerningu), a architektury takie jak VGG lub GoogleNet (według autora zbioru) potrafiły osiągnąć na tym zbiorze 96-98% dokładności.

Przykłady wykres dla sieci typu VGG