# Klasyfikacja gatunków ptaków

Aleksandra Śliwska

listopad 2023 - styczeń 2024

# Tydzień 1

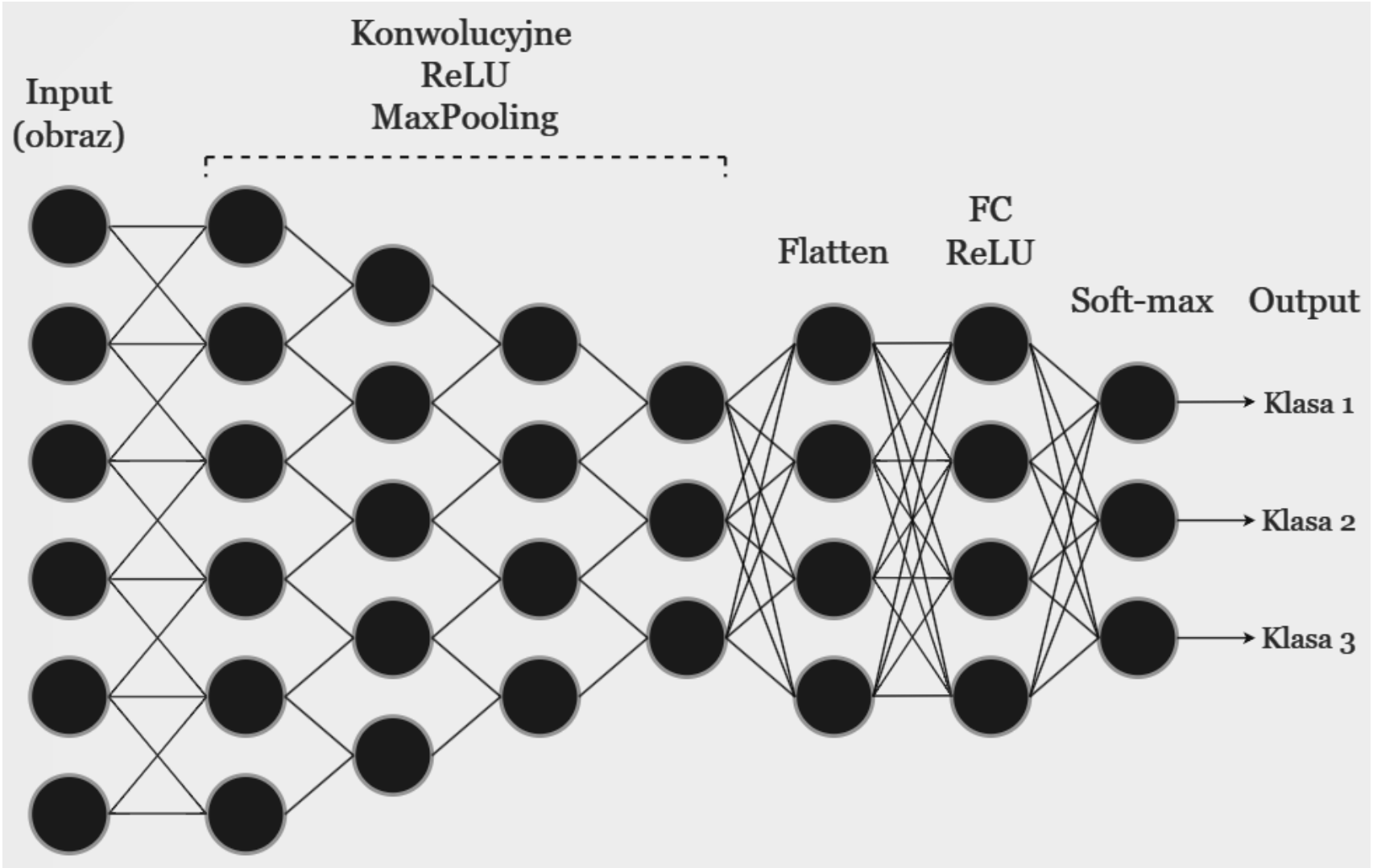
# Dane

Obrazy ptaków: BIRDS 525 SPECIES - IMAGE CLASSIFICATION

*Kwiatownik szkarłatny/Crimson Sunbird*

Wymiary: 224 x 224 x 3 px
525 klas, 90 000 zdjęć
70% trenowanie, 15% walidacja, 15% testowanie

# Model




EfficientNetB0?

# Metryki

Optymalizacja dla **F1 score**

precision & recall - One-vs-Rest, weighted average
accuracy

# Progi i dopuszczalne straty

F1 score $\approx 0.8$

spadek jakości przy kompresji $\approx 1\%$

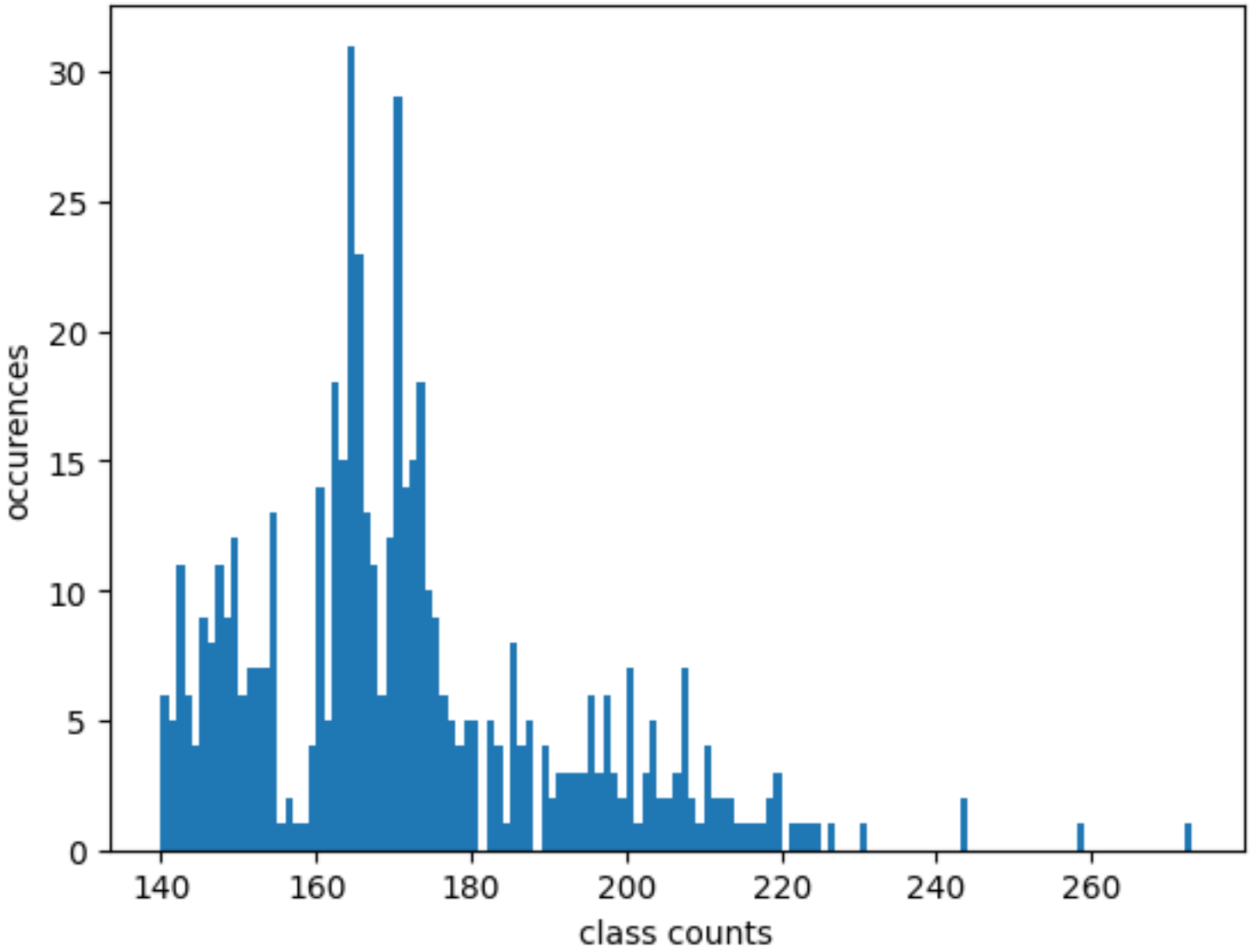
# Tydzień 2

# Naprawa błędów w dataset'cie

a) naprawa nazw w csv i folderach
b) zmiana wymiarów obrazów do (224, 244)
c) dystrybucja plików 70%/15%/15%
d) dodatkowe sprawdzanie poprawności datasetu

| label | train | valid | test | lost count |
|---|---|---|---|---|
| ABBOTTS BABBLER | 70.52 | 15.03 | 14.45 | 0 |
| ABBOTTS BOOBY | 70.00 | 14.74 | 15.26 | 0 |
| ABYSSINIAN GROUND HORNBILL | 70.37 | 14.81 | 14.81 | 0 |
| AFRICAN CROWNED CRANE | 70.34 | 15.17 | 14.48 | 0 |
| AFRICAN EMERALD CUCKOO | 70.12 | 14.63 | 15.24 | 0 |
| AFRICAN FIREFINCH | 70.07 | 14.97 | 14.97 | 0 |

# Rozkład rozmiarów klas



# Pierwsze próby napisania sieci

a) stworzenie klasy do wczytywania danych
b) dataloaders, batch size = 32
c) modyfikacja EfficientNetB0; CrossEntropyLoss, SGD, scheduler.StepLR
b) niskie accuracy (na razie tylko liczenie tej metryki)

# Tydzień 3

# Dodane funkcjonalności

- wagi dla klas, obliczanie weighted f1 score
- walidacja, testowanie
- możliwość ograniczenia ilości klas
- wczesne przerywanie uczenia, zmieniony scheduler
- zapisywanie modelu, konwersja do onnx

# Tuning hiperparametrów

```
model.classifier = nn.Sequential(
    nn.Linear(1280, 900),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(900, 650),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(650, 525)
)
```

- EarlyStopper(patience=2, threshold=0) (monitoruje f1 score)
- ReduceLROnPlateau(factor=0.1, patience=1, threshold_mode="abs") (monitoruje loss)
- learning rate = 3e-3
- weighted CrossEntropyLoss

# Efekty

```
[epoch:  1/50, time: 205s] loss: 5.666, f1 score: 0.033, accuracy: 0.054
[epoch:  2/50, time: 209s] loss: 2.611, f1 score: 0.402, accuracy: 0.425
[epoch:  3/50, time: 209s] loss: 1.717, f1 score: 0.585, accuracy: 0.591
[epoch:  4/50, time: 209s] loss: 1.403, f1 score: 0.656, accuracy: 0.655
[epoch:  5/50, time: 208s] loss: 1.214, f1 score: 0.698, accuracy: 0.695
[epoch:  6/50, time: 210s] loss: 1.103, f1 score: 0.722, accuracy: 0.721
[epoch:  7/50, time: 209s] loss: 1.016, f1 score: 0.742, accuracy: 0.739
[epoch:  8/50, time: 207s] loss: 0.955, f1 score: 0.758, accuracy: 0.757
[epoch:  9/50, time: 207s] loss: 0.910, f1 score: 0.767, accuracy: 0.765
[epoch: 10/50, time: 207s] loss: 0.886, f1 score: 0.772, accuracy: 0.771
[epoch: 11/50, time: 206s] loss: 0.858, f1 score: 0.780, accuracy: 0.778
[epoch: 12/50, time: 209s] loss: 0.861, f1 score: 0.782, accuracy: 0.779
[epoch: 13/50, time: 210s] loss: 0.850, f1 score: 0.784, accuracy: 0.782
[epoch: 14/50, time: 212s] loss: 0.799, f1 score: 0.796, accuracy: 0.793
[epoch: 15/50, time: 212s] loss: 0.777, f1 score: 0.801, accuracy: 0.799
[epoch: 16/50, time: 211s] loss: 0.805, f1 score: 0.797, accuracy: 0.795
[epoch: 17/50, time: 209s] loss: 0.766, f1 score: 0.804, accuracy: 0.802
[epoch: 18/50, time: 208s] loss: 0.742, f1 score: 0.809, accuracy: 0.807
[epoch: 19/50, time: 209s] loss: 0.752, f1 score: 0.808, accuracy: 0.805
[epoch: 20/50, time: 206s] loss: 0.760, f1 score: 0.805, accuracy: 0.803
Epoch 00020: reducing learning rate of group 0 to 3.0000e-04.
[epoch: 21/50, time: 208s] loss: 0.742, f1 score: 0.808, accuracy: 0.806
stopping early

[Test] loss: 0.742, f1 score: 0.816, accuracy: 0.813
```

# Dokończenie sieci i napisanie aplikacji

# Naprawa preprocessingu obrazów

Przy użyciu domyślnych operacji dla EfficientNetB0 zdjęcia były niepotrzebnie powiększane do rozmiarów 256x256 i ponownie przycinane centralnie do 224x224.

```
# preprocess = weights.transforms()
preprocess = v2.Compose([
    v2.ToImage(),
    v2.ToDtype(torch.uint8, scale=True),
    v2.Resize(size=(224, 224), antialias=True),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Naprawa tego nie zmieniła wyników testów, ale przyspieszyła każdą epokę o parę sekund.

# Kwantyzacja

Dodana działająca możliwość kwantyzacji statycznej i konwersji oryginalnego modelu .pt do .ptl.

Niestety model EfficientNetB0 okazał się całkowicie niekwantyzowalny, a próby podmienienia podstawowego modelu na EfficientNetB0_lite wykazały się 4-krotnie dłuższym czasem uczenia i końcowym wynikiem testów $\sim 37\%$.

Końcowa sieć użyta w aplikacji to nieskompresowany model_525_ver3.onnx ważący 23.2 MB, oparty na EfficientNetB0.

# Aplikacja androidowa

Aplikację napisano w języku Kotlin z użyciem oficjalnych bibliotek Android, Jetpack Compose oraz ONNX Runtime - Android.

Do ułatwienia pracy zastosowano także nieoficjalne biblioteki:
- Compose Destinations do uproszczenia kodu nawigującego między ekranami aplikacji,
- Coil do asynchronicznego wyświetlania obrazów przy użyciu ich ścieżek URI.

Wiele funkcji do obsługi ONNX Runtime zapożyczono lub wzorowano na oficjalnych przykładach zastosowania onnxruntime-android do inferencji.

# Wynik końcowy