# WOJSKOWA AKADEMIA TECHNICZNA

**im. Jarosława Dąbrowskiego**

## WYDZIAŁ CYBERNETYKI



# Sprawozdanie

Zaawansowane metody uczenia maszynowego

**Sieci konwolucyjne - AlexNet**


Autor:

**Karol Baranowski**

Prowadzący:

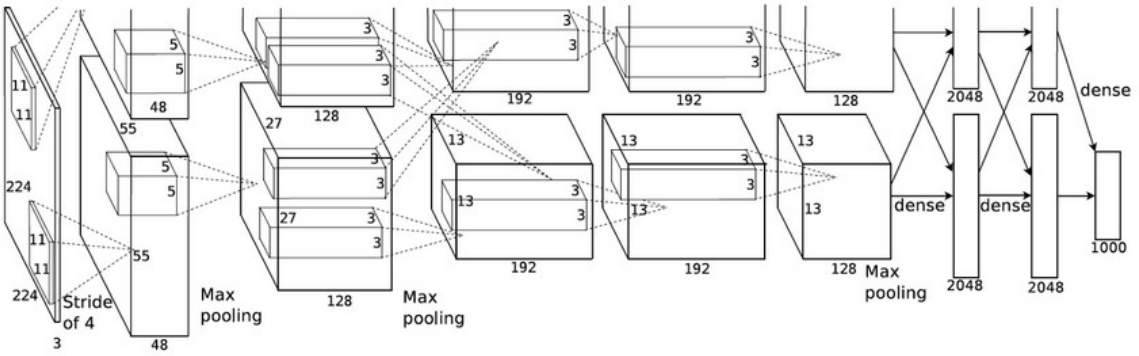
**mgr inż. Przemysław Czuba**

# Spis treści

# Zadanie

- Zbiór danych:
  - CIFAR10
  - https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
  - 60000 32x32 obrazków, 10 klas, 6000 obrazków na klasę (samoloty, samochody, koty...)
- Transformacje zbioru treningowego:
  - Rotacja o losowy kąt (+- 5)
  - Losowe odwrócenie horyzontalne (p = 0.5) (**dlaczego?**)
  - Normalizacja
- Wyrysowanie 10 obrazków dla dowolnie zdefiniowanego filtra
- Użycie BATCH_SIZE = 256 (dlaczego większy batch_size przyspiesza uczenie?)
- Znalezienie **learning rate** używając **LRFinder** i użycie go w algorytmie optymalizacji
  - Inicjalny LR = 1e-7
  - Maxymalny LR = 10
  - Zaprezentować proces poszukiwania (wykres)
  - Jak daleko jest znaleziony LR od defaultowej wartości LR użytego algorytmu?
- Trenowanie modelu
  - Użycie w modelu **nn.Dropout**[2]
- Ewaluacja modelu:
  - Wypisanie **Train** oraz **Test Accuracy**
  - Confusion Matrix
  - Wykres 30 najgorzej zaklasyfikowanych obrazków

# Wstęp




AlexNet ma 7 warstw: 5 konwolucyjnych i 2 w pełni połączone. W implementowanej sieci parametry określono analogicznie jak w powyższym diagramie z tym, że dopasowano ilość neuronów w pierwszej warstwie, aby odpowiadały obrazkom z CIFAR10, które mają 32 x 32 px, nie 28 x 28 jak obrazki z MNIST. Główna część modelu wygląda w związku z tym następująco:

```
"""
Model AlexNet
"""
class AlexNet(nn.Module):
  # output_dim - liczba klas
  def __init__(self, output_dim):
    super().__init__()

    self.features = nn.Sequential(
        nn.Conv2d(3, 64, 3, 2, 1),
        nn.MaxPool2d(2), # wielkość filtra
        nn.ReLU(inplace = True),
        nn.Conv2d(64, 192, 3, padding = 1),
        nn.MaxPool2d(2),
        nn.ReLU(inplace = True),
        nn.Conv2d(192, 384, 3, padding = 1),
        nn.ReLU(inplace = True),
        nn.Conv2d(384, 256, 3, padding = 1),
        nn.ReLU(inplace = True),
        nn.Conv2d(256, 256, 3, padding = 1),
        nn.MaxPool2d(2),
        nn.ReLU(inplace = True)
    )

    self.classifier = nn.Sequential(
        nn.Dropout(0.5),
        nn.Linear(256 * 2 * 2, 4096),
        nn.ReLU(inplace = True),
        nn.Dropout(0.5),
        nn.Linear(4096, 4096),
        nn.ReLU(inplace = True),
        nn.Linear(4096, output_dim),
    )

  def forward(self, x):
    x = self.features(x)
    h = x.view(x.shape[0], -1)
    x = self.classifier(h)
    return x, h

# Stworzenie modelu z 10 klasami wyjściowymi
model = AlexNet(10)
```

Loading...

Na wyjściu ustawiono 10 neuronów jako 10 klas wyjściowych. Pozostałą część kodu implementowano z użyciem funkcji z laboratorium 4 (LeNet), przekształcając je odpowiednio do wymogów AlexNet i danych CIFAR10. Użyto również runtime GPU na Colab, ponieważ na maszynie autora jedna epoka uczenia trwała ok. 40 min.

## 1. Zbiór danych:

60000 32x32 obrazków, 10 klas, 6000 obrazków na klasę (samoloty, samochody, koty…)

Dane udało się poprawnie wczytać i rozdzielić na zbiór treningowy, walidacyjny i testowy:

```
Files already downloaded and verified
Files already downloaded and verified
Number of training examples: 45000
Number of validation examples: 5000
Number of testing examples: 10000
```

## 2. Transformacje zbioru treningowego:

Rotacja o losowy kąt (+- 5), losowe odwrócenie horyzontalne (p = 0.5) (dlaczego?), normalizacja:

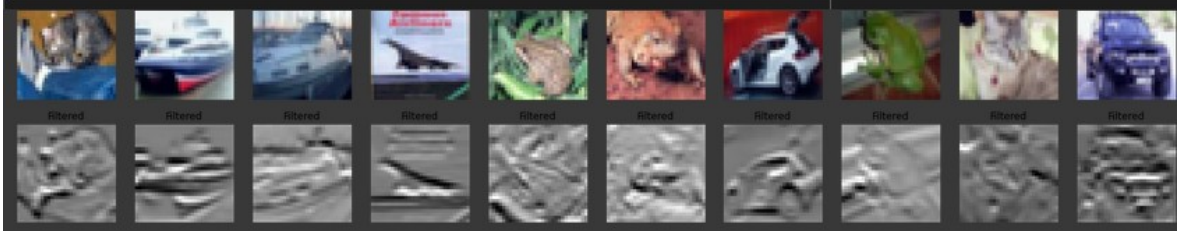
```
train_transforms = transforms.Compose([
  transforms.RandomRotation(5),
  transforms.RandomCrop(32, padding=2),
  transforms.RandomHorizontalFlip(0.5),
  transforms.ToTensor(),
  transforms.Normalize(mean=mean, std=std)
])
```

Dodatkowe odwrócenie horyzontalne z prawdopodobieństwem 0.5 odbije lustrzanie połowę obrazków, czyniąc sieć lepiej wytrenowaną po nauce na takim zbiorze, ponieważ obiekty rzeczywiste mogą być odwrócone. Dla cyfr z MNIST nie miało to sensu.

## 3. Wyrysowanie 10 obrazków dla dowolnie zdefiniowanego filtra

10 obrazków dla filtru horyzontalnego z laboratorium 4:

```
N_IMAGES = 10
images = [image for image, label in [test_data[i] for i in range(N_IMAGES)]]
horizontal_filter = [[-1, -2, -1],
                     [0, 0, 0],
                     [1, 2, 1]]
plot_filter(images, horizontal_filter)
```




Na filtrze widać, że poziome krawędzie obiektów są pogrubione na czarno

## 4. Użycie BATCH_SIZE = 256 (dlaczego większy batch_size przyspiesza uczenie?)

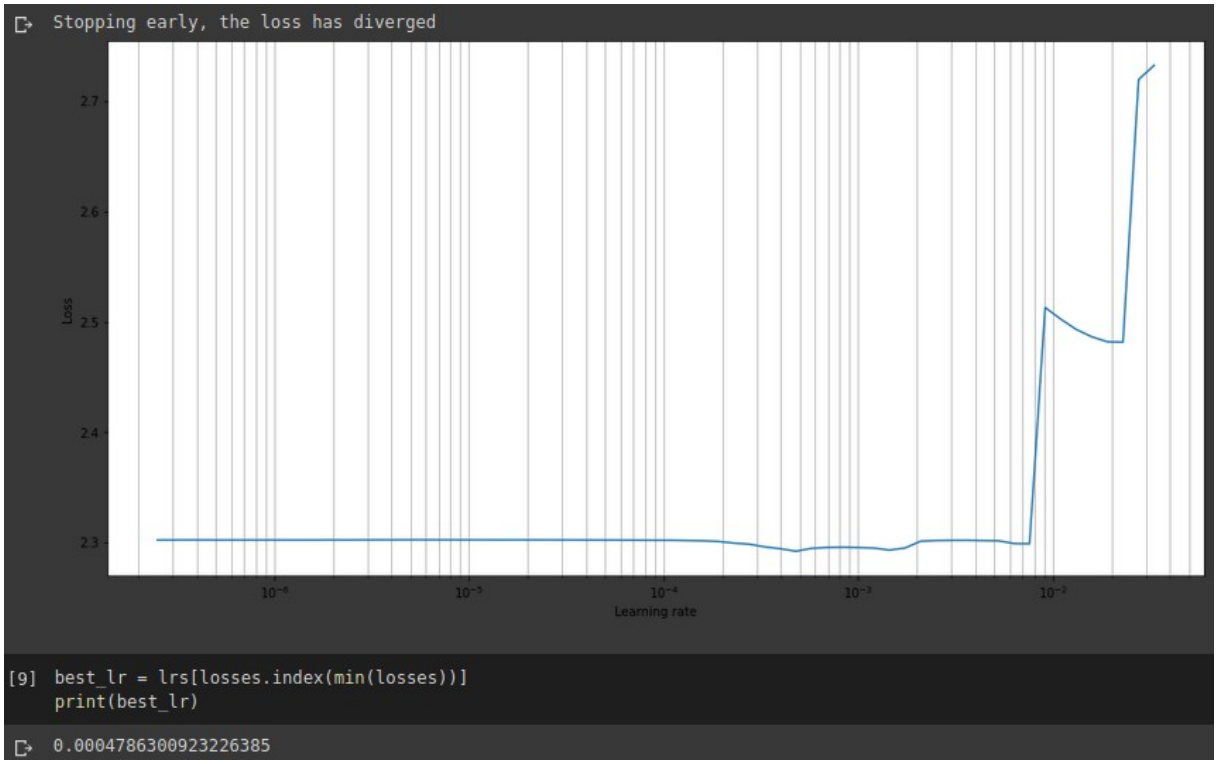
Batch size to ilość danych (obrazków) pobieranych jednocześnie. AlexNet ma bardzo dużo parametrów i obrazki z CIFAR10 są znacznie większe niż w MNIST i mają 3 kanały koloru, więc ich przetworzenie trwa dłużej. Używając wielkości 256 zamiast 64, pobieramy więcej parametrów naraz co przyśpieszy uczenie.

## 5. Znalezienie learning rate używając LRFinder i użycie go w algorytmie optymalizacji

Inicjalny LR = 1e-7 (start), maxymalny LR = 10 (stop):

```
# Oszacowanie learning rate
start = 1e-7
optimizer = optim.Adam(model.parameters(), lr = start)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
criterion = nn.CrossEntropyLoss()

model = model.to(device)
criterion = criterion.to(device)
stop = 10 # skończ, gdy learning rate dojdzie do 10
iteration = 100
```




Zaprezentować proces poszukiwania (wykres):

Jak daleko jest znaleziony LR od defaultowej wartości LR użytego algorytmu?

Użyty algorytm optymalizacyjny Adam domyślnie używa learning rate równe 0.001. Znaleziony learning rate to ~0.0005, a więc mniejszy o 0.0005. Można uznać, że wartości są zbliżone i wyliczony learning rate powinien dać dobre efekty. Na wykresie widać, że koszt utrzymuje się ciągle na niskim poziomie, między 1e-4 i 1e-3 maleje i to tam znaleziono learning rate, a tuż przed 1e-2 drastycznie rośnie.

## 6. Trenowanie modelu

Użycie w modelu nn.Dropout:

Dropout używa się, aby model nie był skutecznie wyszkolony tylko dla zbioru treningowego. Polega to na losowym ustawieniu pewnej części neuronów w warstwie początkowej na 0. Dodaje to nieregularność. Neurony w następnej warstwie uczą się z mniejszej ilości danych (tylko tych, którym nie ustawiono 0). Dzięki temu model nie będzie się dopasowywał do danych treningowych. Przyjmuje się, aby dropout ustawić na 0,5, czyli wyzerować połowę neuronów.

## 7. Ewaluacja modelu:

Wypisanie Train oraz Test Accuracy

Train accuracy po 20 epokach:

```
Epoch: 16 | Epoch Time: 0m 22s
        Train Loss: 0.567 | Train Acc: 80.29%
         Val. Loss: 0.643 |  Val. Acc: 76.97%
Epoch: 17 | Epoch Time: 0m 22s
        Train Loss: 0.544 | Train Acc: 80.97%
         Val. Loss: 0.670 |  Val. Acc: 76.89%
Epoch: 18 | Epoch Time: 0m 22s
        Train Loss: 0.531 | Train Acc: 81.64%
         Val. Loss: 0.626 |  Val. Acc: 78.20%
Epoch: 19 | Epoch Time: 0m 22s
        Train Loss: 0.507 | Train Acc: 82.32%
         Val. Loss: 0.679 |  Val. Acc: 77.45%
Epoch: 20 | Epoch Time: 0m 22s
        Train Loss: 0.488 | Train Acc: 82.88%
         Val. Loss: 0.670 |  Val. Acc: 77.08%
```

Train accuracy wyniosła 82.88%, Validation Accuracy 77.08%, więc mimo dropoutu część zależności została wyuczona ze zbioru treningowego. Natomiast różnica jest mała, a wynik jest zadowalający dla tak zróżnicowanych obrazków

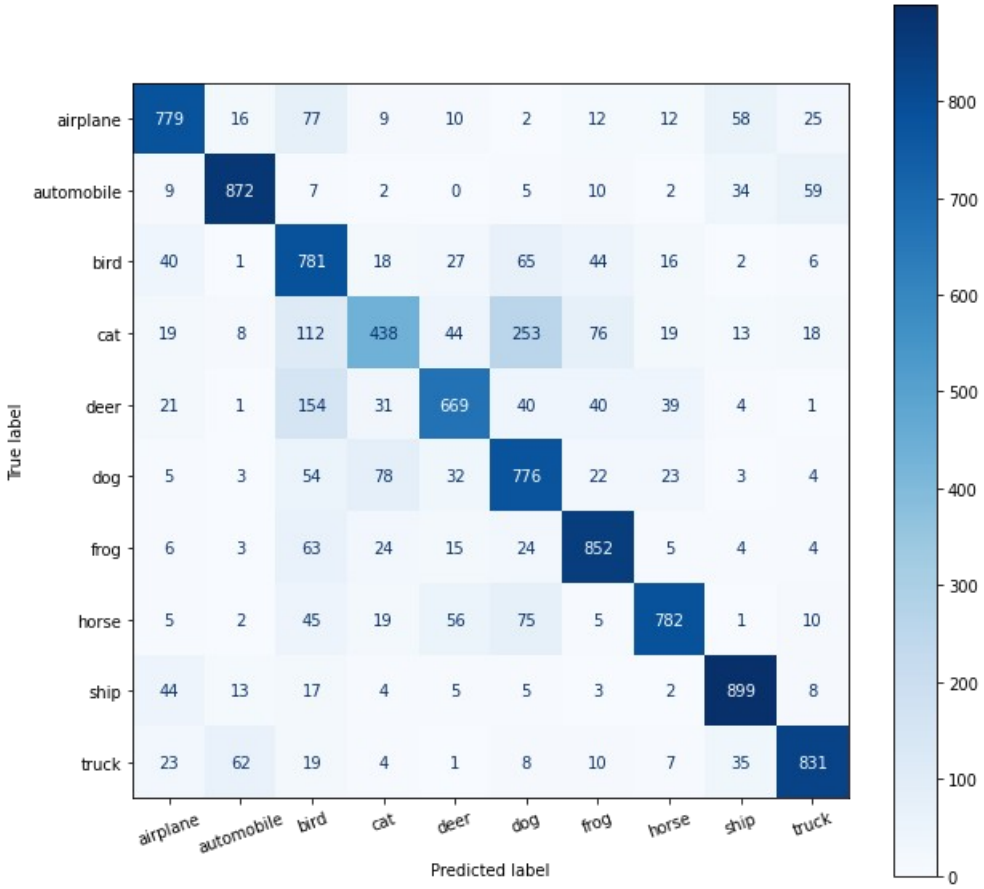
Test accuracy:

```
[12] test_loss, test_acc = evaluate(model, test_iterator, criterion, device)
     print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc * 100:.2f}%')

     Test Loss: 0.695 | Test Acc: 76.89%
```

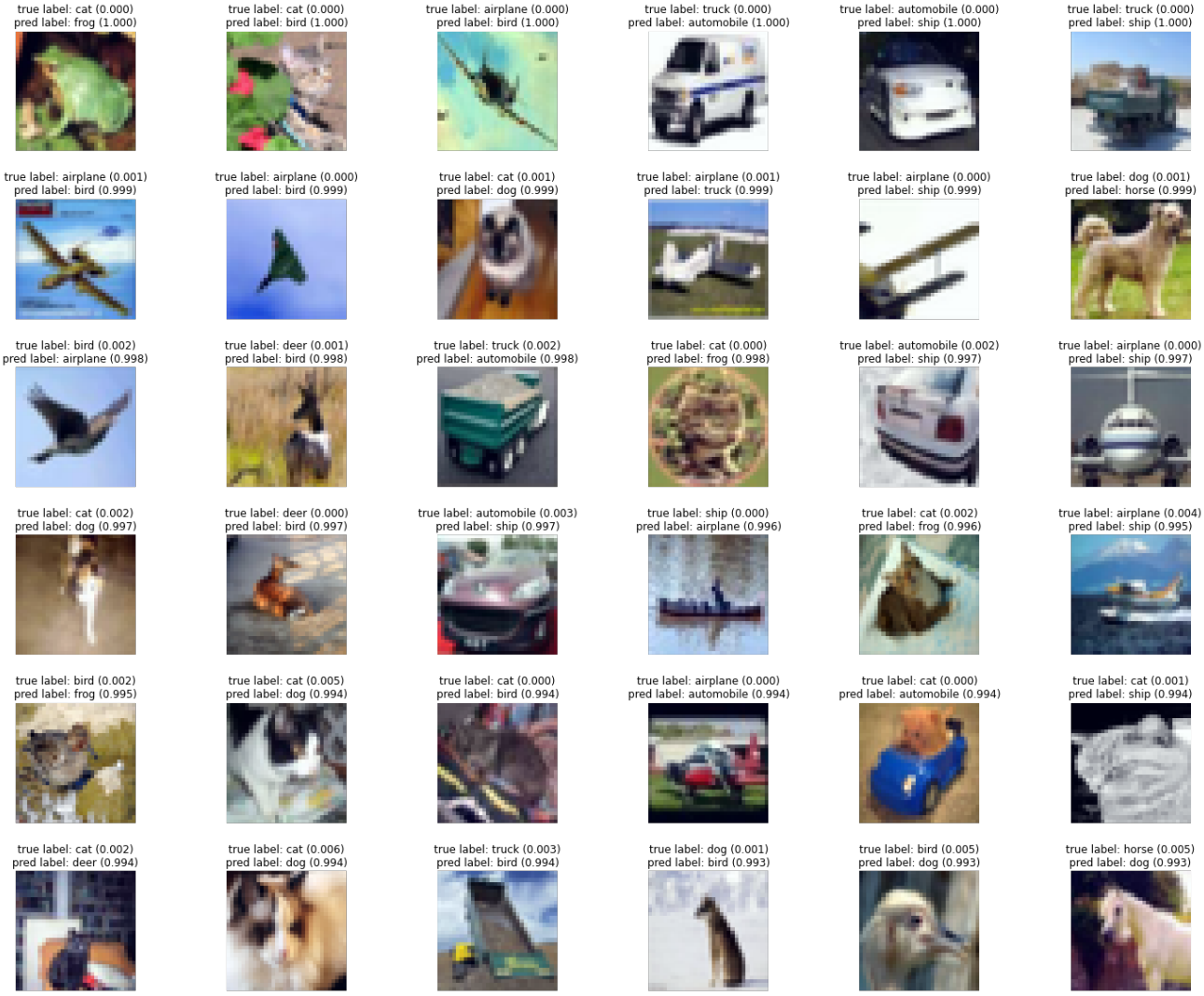
Test accuracy wyniosło 76,89%.

Confusion Matrix:




Widać, że najwięcej pomyłek było psa z kotem i ptaka z sarną.

Wykres 30 najgorzej zaklasyfikowanych obrazków




Widać, że zbiór CIFAR10 ma w sobie podchwytliwe obrazki np. koty przebrane za żabę, kota jadącego samochodem itp. Jednak większość najbardziej pomylonych obrazków jest prosta do rozpoznania dla człowieka. Trzeba natomiast mieć na względzie, że model tylko minimalizuje funkcję kosztu dla klas, które rozpoznaje jako pewne kształty, więc z punktu widzenia modelu kształt samolotu i ptaka jest podobny, tak samo jak kota i psa.

## Wnioski

Cel zadania został osiągnięty. Wszystkie polecenia udało się wykonać i zaimplementować sieć AlexNet, której dokładność oszacowano na 76% po 20 epokach uczenia. Na tak rozbudowany, różnorodny i podchwytliwy zbiór danych jak CIFAR10 rozpoznanie 3 z 4 obrazków można uznać za sukces, natomiast widać tu barierę i prawdopodobnie trzeba użyć/opracować lepszy model aby zwiększyć dokładność.