

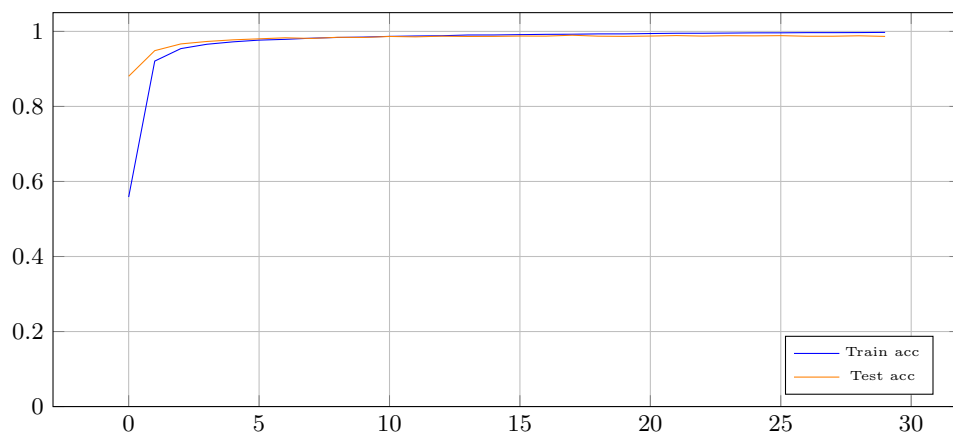
# 1 Eksperyment 1: Architektury sieci spłotowej dla MNIST (JakubPawlak)

## Pierwsza architektura

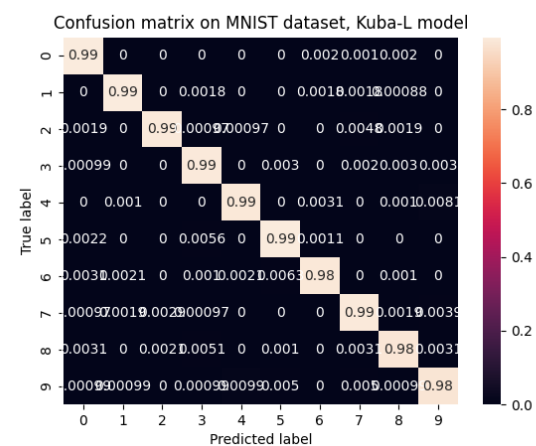
Pierwsza architektura składa się z warstwy spłotowej z jądrem  $5 \times 5$ , na wyjściu której obraz ma 6 kanałów, następnie jest przepuszczony przez funkcje sigmoid oraz max pooling. Zamierzeniem jest, aby ta warstwa wykrywała krawędzie lub rogi cyfr. Z tego też powodu, użyto funkcji aktywacji sigmoid, ponieważ funkcja ReLU całkowicie zeruje ujemne wartości. W przypadku wykrywania krawędzi, ujemne wartości mogą być używane do reprezentacji kierunku.

Drugi spłot prowadzi do 16 kanałów i ponownie jest używana funkcja sigmoid oraz max pooling. Zadaniem tej warstwy jest rozpoznanie większych, bardziej złożonych kształtów. Finalnie ekstraktor prowadzi do tensora  $16 \times 5 \times 5$ , który zostaje spłaszczony.

Klasyfikator to MLP z rozmiarami warstw odpowiednio 400, 120, 84, 10, używający sigmoidy jako funkcji aktywacji.



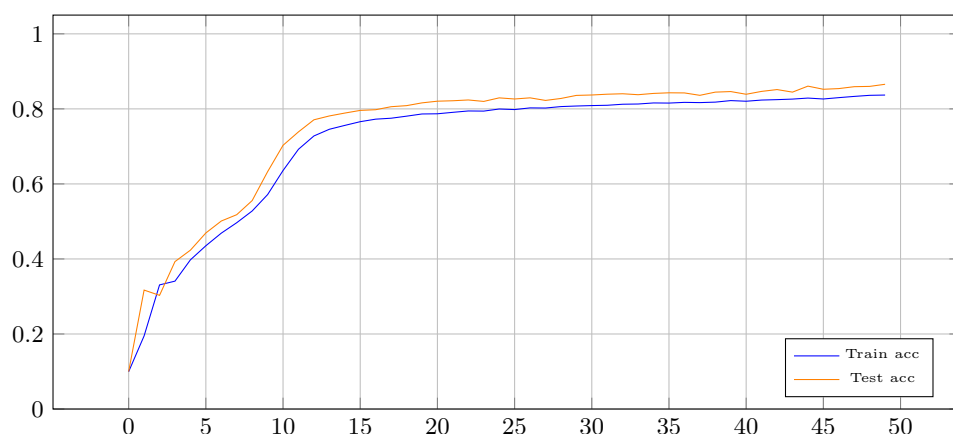
(a) Wykres zmian accuracy



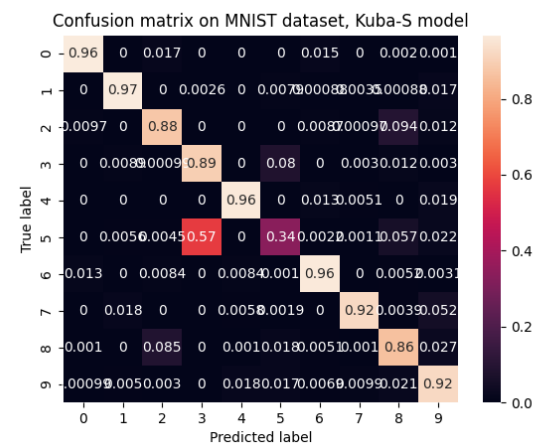
(b) Macierz pomyłek

Rysunek 1: Wyniki dla 1 architektury (Train acc: 0.997; Test acc: 0.987)

## Druga architektura, prowadząca do ekstrakcji 2 cech



(a) Wykres zmian accuracy



(b) Macierz pomyłek

Rysunek 2: Wyniki dla 2 architektury (Train acc: 0.836; Test acc: 0.866)

W drugiej architekturze warunkiem koniecznym jest rozmiar wektora cech, wynoszący 2 cechy. Jest to umotywowane chęcią stworzenia wizualizacji granic decyzyjnych. Ekstraktor cech składa się z następujących warstw:

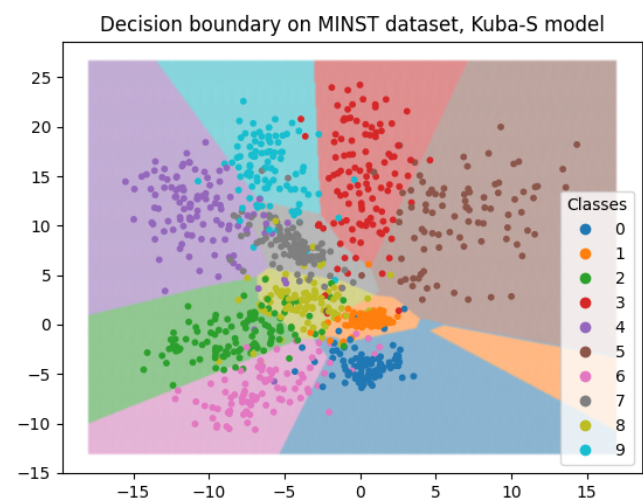
```
(feature_extractor): Sequential(
  (0): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU()
  (2): Dropout2d(p=0.1, inplace=False)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(10, 5, kernel_size=(5, 5), stride=(1, 1))
  (5): ReLU()
  (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (7): Conv2d(5, 2, kernel_size=(3, 3), stride=(1, 1))
  (8): ReLU()
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=18, out_features=2, bias=True)
)
(classifier): Sequential(
  (0): Linear(in_features=2, out_features=5, bias=True)
  (1): ReLU()
  (2): Linear(in_features=5, out_features=10, bias=True)
)
```

W tym modelu, aby jeszcze bardziej zmniejszyć ilość cech, zastosowano jeszcze jedną warstwę konwolucyjną. Finalnie, ekstraktor cech zawiera jedną w pełni połączoną warstwę, w celu redukcji do 2 cech.

W tym modelu postanowiono również przeprowadzić eksperyment z użyciem warstwy dropout w celu poprawienia odporności na overfitting. Pomimo, że z racji na dużą liczbę zbioru treningowego, w eksperymencie 1 nie ma dużego ryzyka przetrenowania, sytuacja może ulec zmianie w eksperymencie 2. Negatywnym efektem użycia warstwy dropout jest spowolnienie procesu uczenia, co jest dobrze widoczne na wykresie.

Warstwy konwolucyjne prowadzą do finalnego tensora o wymiarach  $2 \times 3 \times 3$ , 10 więc na końcu ekstraktora cech znajduje się jeszcze jedna warstwa linear, tworząca 2-wymiarowy wektor cech.

Klasyfikator zawiera jedną warstwę ukrytą o rozmiarze 5 neuronów, z funkcją relu, a następnie warstwę wyjściową o rozmiarze 10 neuronów.



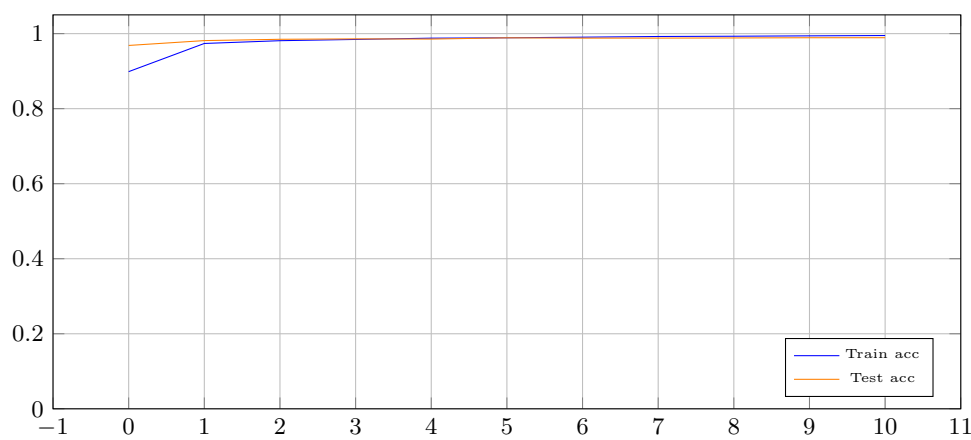
Rysunek 3: Granica decyzyjna dla arch. 2

## 2 Eksperyment 1: Architektury sieci spłotowej dla MNIST (Magdalena Pakuła)

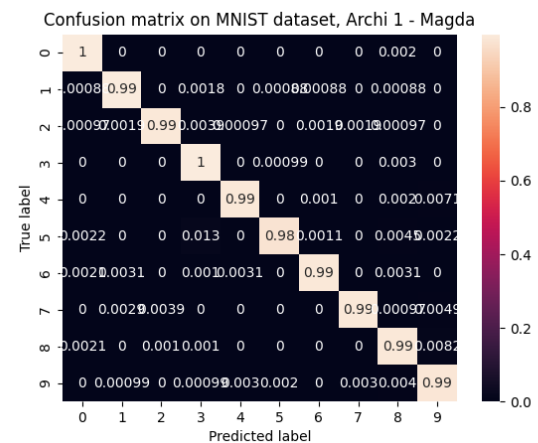
### Pierwsza architektura

Zawiera trzy warstwy spłotowe każda warstwa wydobywa coraz bardziej abstrakcyjne cechy. Warstwy początkowe rejestrują krawędzie i tekstury, podczas gdy warstwy głębsze rejestrują kształty i bardziej złożone wzory. Ma to kluczowe znaczenie w przypadku rozpoznawania cyfr, gdzie należy rozróżnić subtelne różnice. Po każdej warstwie spłotowej następuje redukcja wymiarowości, dzięki użyciu MaxPool2d, co pozwala sieci skupić się na najbardziej istotnych cechach cyfr, poprawiając generalizację. Stopniowy wzrost filtrów (8, 16, 32) zapewnia równowagę między wydajnością obliczeniową a bogactwem funkcji. Ostatecznie model zostaje spłaszczony do 288 obiektów, co jest wystarczającą pojemnością do reprezentowania złożonych wzorów, nie będąc jednocześnie zbyt dużym, co mogłoby prowadzić do nadmiernego dopasowania.

Rezultaty tej metody zostały przedstawione poniżej na wykresie accuracy, macierzy pomyłek i wyników accuracy dla najlepszego modelu na rys. 4.



(a) Wykres zmian accuracy



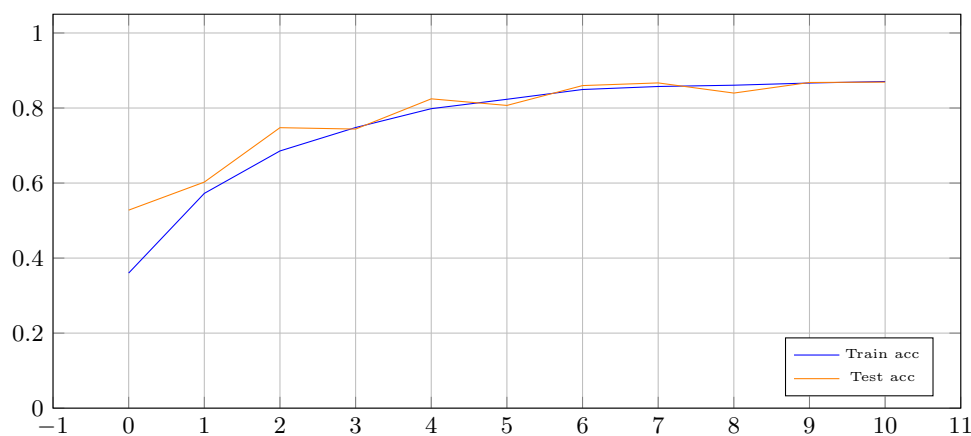
(b) Macierz pomyłek

Rysunek 4: Wyniki dla 1 architektury (Train acc: 0.988; Test acc: 0.984)

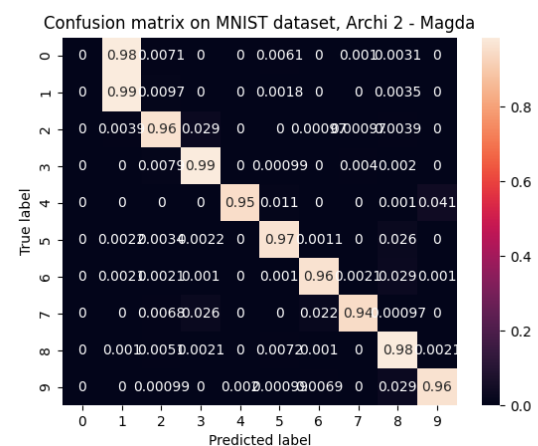
### Druga architektura, prowadząca do ekstrakcji 2 cech

Skląda się z czterech warstw spłotowych. Liczba filtrów w tych warstwach jest hierarchicznie zwiększana (32,64,128) kończąc na 2. Po każdej warstwie spłotowej następuje normalizacja wsadowa (BatchNorm2d), funkcja aktywacji ReLU i redukcja wymiarowości poprzez użycie operacji MaxPool2d, z wyjątkiem ostatniej warstwy, gdzie ta operacja nie jest już potrzebna. Taka konfiguracja została zaprojektowana w celu stopniowego wyodrębniania coraz bardziej skomplikowanych cech z obrazów MNIST. Pierwsze warstwy koncentrują się na wykrywaniu podstawowych cech, takich jak krawędzie i rogi, dzięki mniejszej liczbie filtrów. Następnie, w kolejnych warstwach, liczba filtrów jest stopniowo zwiększana. Operacje normalizacji wsadowej pomagają w stabilizacji uczenia poprzez zapewnienie jednolitego zakresu wartości na wyjściu z każdej warstwy, co może przyspieszyć proces uczenia oraz poprawić ogólną wydajność sieci. Funkcja aktywacji ReLU została wybrana z powodu jej skuteczności w eliminowaniu zjawiska zanikającego gradientu i aktywowania jedynie istotnych cech.

Rezultaty tej metody zostały przedstawione poniżej na rys. 5.



(a) Wykres zmian accuracy



(b) Macierz pomyłek

Rysunek 5: Wyniki dla 2 architektury (Train acc: 0.871; Test acc: 0.869)

Ekstraktor cech składa się z następujących warstw:

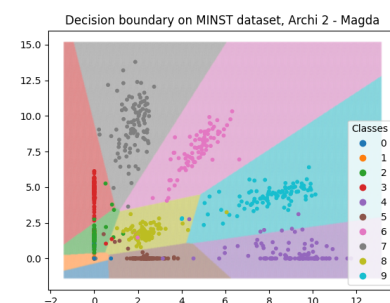
```
(feature_extractor): Sequential(
  (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(32)
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2) # 32x28x28 -> 32x14x14

  (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(64)
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2)

  (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): BatchNorm2d(128)
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2)

  (12): Conv2d(128, 2, kernel_size=(3, 3))
  (13): BatchNorm2d(2)
  (14): ReLU()
  (15): Flatten(start_dim=1)
)
```

```
(classifier): Sequential(
  (0): Linear(in_features=2, out_features=5, bias=True)
  (1): ReLU()
  (2): Linear(in_features=5, out_features=10, bias=True)
)
```

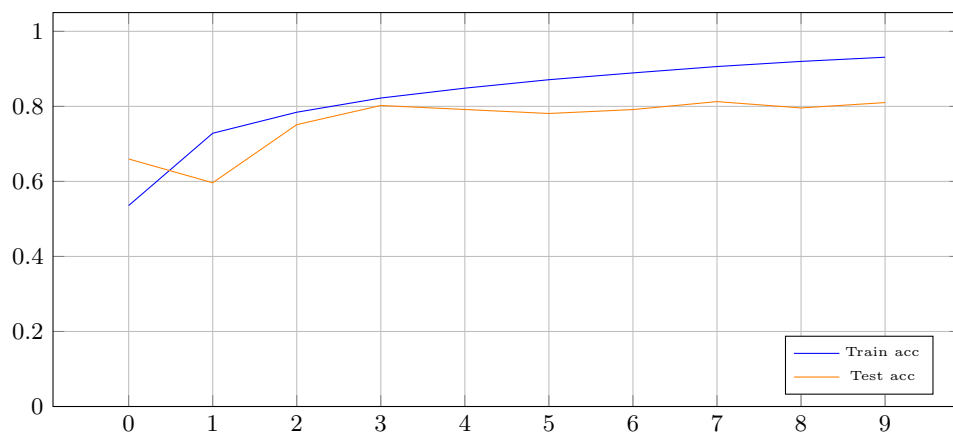


Rysunek 6: Granica decyzyjna dla arch. 2

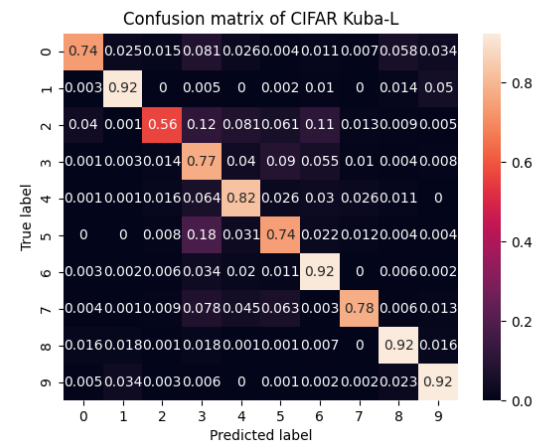
### 3 Eksperyment 1: Architektury sieci spłotowej dla CIFAR10 (Jakub Pawlak)

#### Pierwsza architektura

W tym modelu poczyniono eksperyment z połączeniami rezydualnymi. Dodawanie więcej warstw spłotowych nie zawsze prowadzi do polepszenia skuteczności modelu. Połączenia rezydualne stanowią rozwiązanie tego problemu, ponieważ model może wytrenować się w taki sposób, aby pomijać niepotrzebne warstwy. W celu normalizacji danych zastosowano również warstwę batchnorm2d.



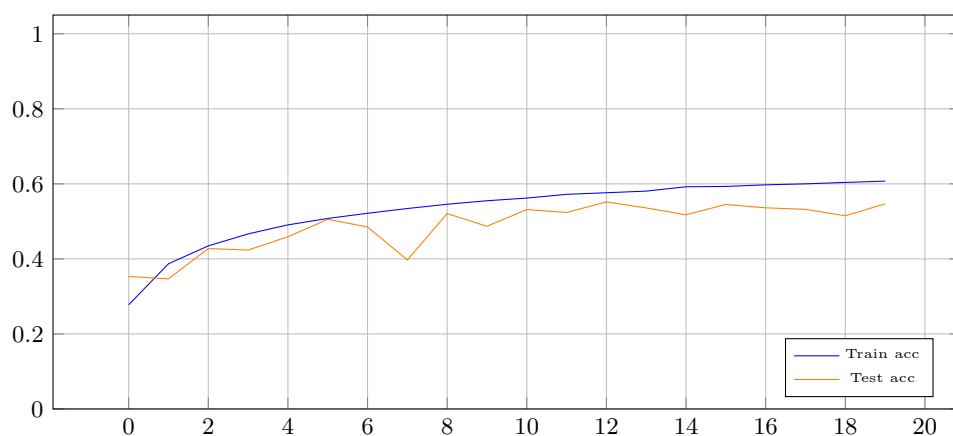
(a) Wykres zmian accuracy



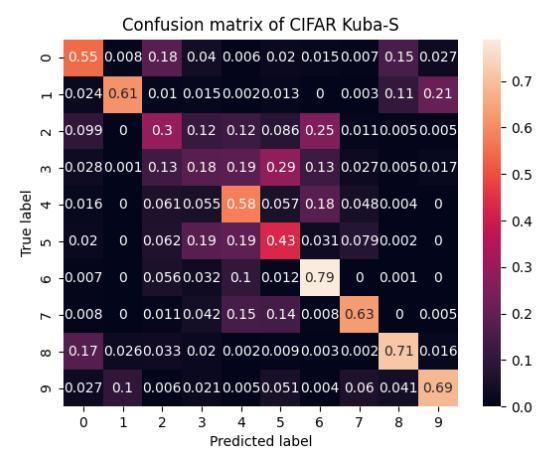
(b) Macierz pomyłek

Rysunek 7: Wyniki dla 1 architektury (Train acc: 0.931; Test acc: 0.810)

#### Druga architektura prowadząca do ekstrakcji 2 cech



(a) Wykres zmian accuracy



(b) Macierz pomyłek

Rysunek 8: Wyniki dla 2 architektury (Train acc: 0.607; Test acc: 0.547)

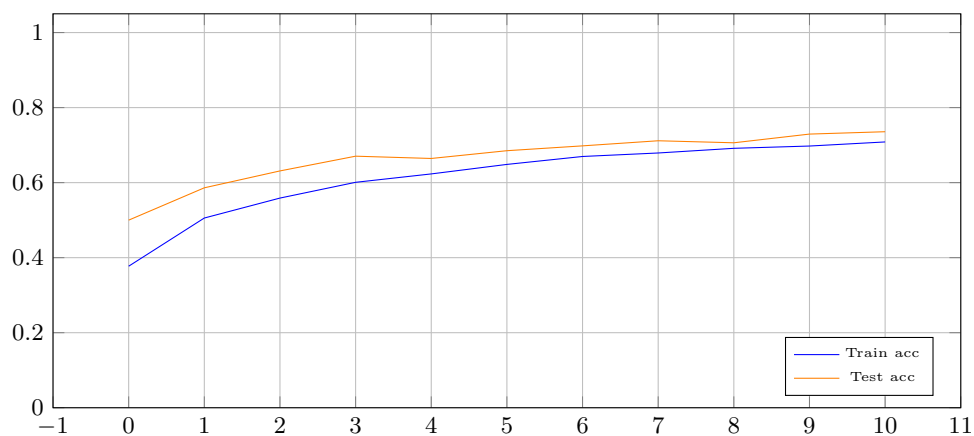
Druga architektura, podobnie jak przy zbiorze MNIST, dokonuje ekstrakcji cech do 2-elementowego wektora. W przeciwieństwie do poprzedniej architektury, ta nie zawiera połączeń rezydualnych, jednak wciąż zawiera warstwę batchnorm2d.

```
(feature_extractor): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(32, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): ReLU()
  (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (11): Conv2d(8, 2, kernel_size=(4, 4), stride=(1, 1))
  (12): Flatten(start_dim=1, end_dim=-1)
)
(classifier): Sequential(
  (0): Linear(in_features=2, out_features=16, bias=True)
  (1): ReLU()
  (2): Linear(in_features=16, out_features=10, bias=True)
)
```

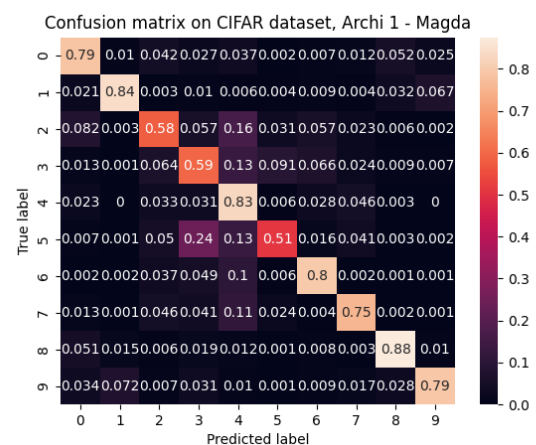
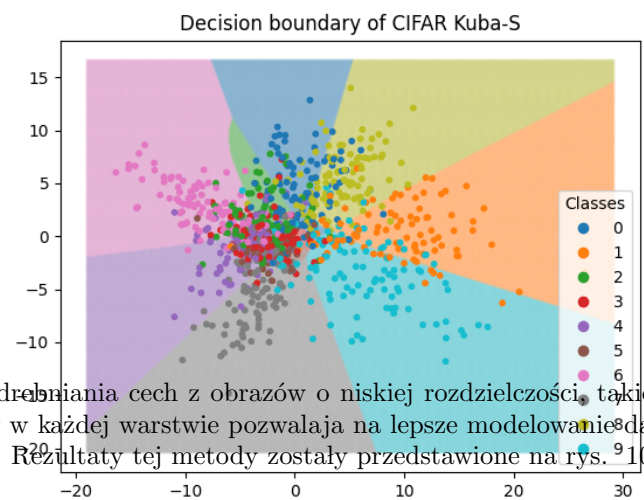
## 4 Eksperyment 1: Architektury sieci spłotowej dla CIFAR10 (Magdalena Pakuła)

### Pierwsza architektura

Składa się z trzech warstw spłotowych, gdzie każda z nich wykorzystuje coraz większą liczbę filtrów o rozmiarze 3x3. Pierwsza warstwa spłotowa zawiera 32 filtry, stosuje funkcje aktywacji ReLU oraz warstwę Dropout z współczynnikiem 0.2, która losowo wyłącza część neuronów, zapobiegając nadmiernemu dopasowaniu. Następnie warstwa MaxPooling zmniejsza rozmiar przestrzenny danych o połowę. Druga warstwa spłotowa składa się z 64 filtrów, ponownie stosuje ReLU oraz Dropout z współczynnikiem 0.3, a także warstwę MaxPooling, która zmniejsza rozmiar przestrzenny danych do 8x8. Trzecia warstwa spłotowa zawiera 128 filtrów, stosuje ReLU, Dropout z współczynnikiem 0.4 oraz MaxPooling, zmniejszając rozmiar przestrzenny danych do 4x4. Dane są następnie spłaszczone do jednowymiarowego tensora za pomocą warstwy Flatten, umożliwiając przekazanie ich do warstw w pełni połączonych w celu klasyfikacji. Klasyfikator składa się z dwóch warstw w pełni połączonych, gdzie pierwsza redukuje rozmiar cech do 128, stosując funkcje ReLU oraz Dropout z współczynnikiem 0.5, a druga warstwa klasyfikuje dane do liczby klas. Ta architektura została zoptymalizowana w celu efektywnego wydobycia cech z obrazów o niskiej rozdzielczości, takich jak te występujące w zbiorze CIFAR. Większa liczba warstw spłotowych oraz zwiększenie liczby filtrów w każdej warstwie pozwalają na lepsze modelowanie danych, jednak możemy też zauważyć dość niskie train accuracy ze względu na ilość użytych funkcji Dropout. Wyniki tej metody zostały przedstawione na rys. 10.



(a) Wykres zmian accuracy



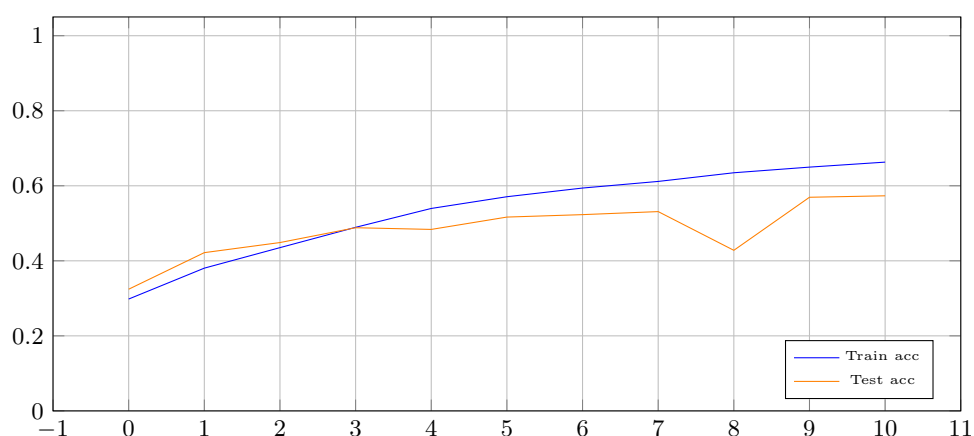
(b) Macierz pomyłek

Rysunek 10: Wyniki dla 1 architektury (Train acc: 0.623; Test acc: 0.679)

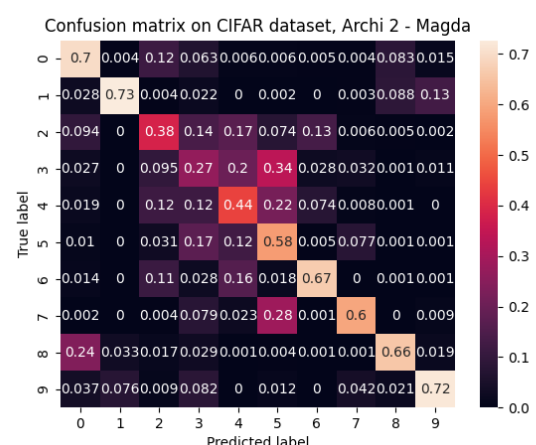
### Druga architektura prowadząca do ekstrakcji 2 cech

Składa się z 4 warstw spłotowych z 32, 64 i 128 filtrami, które pozwalają na stopniowe wydobycie coraz bardziej złożonych cech z danych wejściowych. Następnie użycie BatchNorm2d stabilizuje proces trenowania i pozwala na szybsze uczenie się modelu, a funkcja aktywacji ReLU wprowadza nieliniowość. Następnie jest warstwa MaxPooling redukująca rozmiar danych. Ostatnia warstwa spłotowa prowadzi do 2 filtrów. Na końcu dodana jest funkcja aktywacji ReLU. Klasyfikator składa się z trzech warstw w pełni połączonych, które stopniowo redukują rozmiar cech i umożliwiają klasyfikację danych do odpowiedniej liczby klas. Dropout (0.3) w pierwszej warstwie w pełni połączonej zapobiega nadmiernemu dopasowaniu modelu poprzez losowe wyłączenie neuronów podczas trenowania. Możemy dostrzec różnice w klasyfikacji modelu, a użyciem funkcji Dropout w obu architekturach.

Wyniki przedstawione są na rys. 11.



(a) Wykres zmian accuracy



(b) Macierz pomyłek

Rysunek 11: Wyniki dla 2 architektury (Train acc: 0.539; Test acc: 0.573)

```
(feature_extractor): Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(32)
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2)

  (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(64)
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2)

  (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): BatchNorm2d(128)
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2)

  (12): Conv2d(128, 2, kernel_size=(4, 4))
  (13): ReLU()
  (14): Flatten(start_dim=1)
)

(classifier): Sequential(
  (0): Linear(in_features=2, out_features=128, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.3)
  (3): Linear(in_features=128, out_features=64, bias=True)
  (4): ReLU()
  (5): Linear(in_features=64, out_features=num_classes, bias=True)
)
```

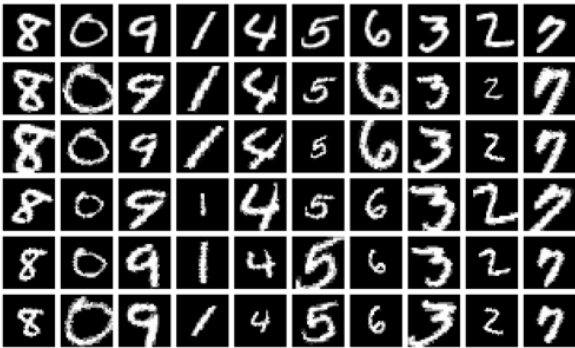


## 5 Eksperyment 2: Wyniki dla MNIST

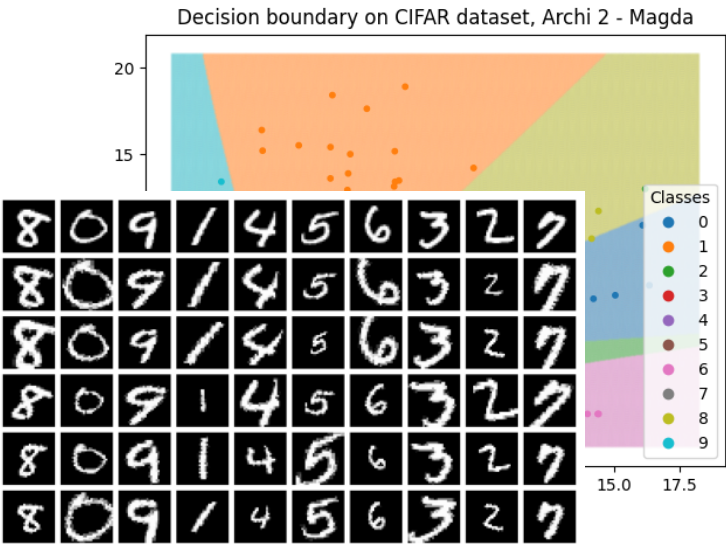
### Augmentacje

Augmentacje:

1. `v2.RandomAffine(degrees=10, scale=(0.6, 1.5), shear=20)` — Jakub Pawlak
2. `v2.ElasticTransform(alpha=50.0, sigma=5.0)` — Magdalena Pakuła



(a) Jakub Pawlak



Rysunek 12: Granica decyzyjna dla arch. 2

Rysunek 13: Augmentacje dla zbioru MNIST (górny wiersz zawiera oryginalne obrazki)

### Najlepsza architektura

Architektura Kuba

Augmentacja	Ilość obrazów		
	10	200	1000
Jakub Pawlak	$0.100 \pm 0.001$	$0.795 \pm 0.042$	$0.921 \pm 0.024$

Tabela 1: Wyniki dla 1 architektury na zbiorze MNIST z augmentacją (accuracy z 10 prób w formie  $\mu \pm \sigma$ )

### Najlepsza architektura prowadzaca do ekstrakcji 2 cech

Architekrura Magda

Augmentacja	Ilość obrazów		
	10	200	1000
Jakub Pawlak	$0.106 \pm 0.031$	$0.385 \pm 0.037$	$0.530 \pm 0.045$
Magdalena Pakuła	$0.134 \pm 0.041$	$0.375 \pm 0.054$	$0.533 \pm 0.039$

Tabela 2: Wyniki dla 2 architektury na zbiorze MNIST z augmentacją (accuracy z 10 prób w formie  $\mu \pm \sigma$ )

## 6   Eksperyment 2: Wyniki dla CIFAR10

### Augmentacje



(a) Jakub Pawlak



(b) Magdalena Pakuła

Rysunek 14: Augmentacje dla zbioru CIFAR10 (górny wiersz zawiera oryginalne obrazki)

### Najlepsza architektura

Arch. Kuba

Augmentacja	Ilość obrazów		
	10	200	1000
Jakub Pawlak	$0.126 \pm 0.014$	$0.298 \pm 0.021$	$0.466 \pm 0.029$

Tabela 3: Wyniki dla 1 architektury na zbiorze MNIST z augmentacją (accuracy z 10 prób w formie  $\mu \pm \sigma$ )

### Najlepsza architektura prowadząca do ekstrakcji 2 cech

Arch. Magda

Augmentacja	Ilość obrazów		
	10	200	1000
Jakub Pawlak	$0.100 \pm 0.001$	$0.100 \pm 0.000$	$0.137 \pm 0.046$

Tabela 4: Wyniki dla 2 architektury na zbiorze MNIST z augmentacją (accuracy z 10 prób w formie  $\mu \pm \sigma$ )

## 7 Analiza i wnioski

### Porównanie architektur sieci splotowych

Generalne wnioski Magdy dla MNIST: wyszło zajebiscie, rezultaty same pokazują — można się bić :) Generalne wnioski Magdy dla CIFAR: nie wyszło zajebiscie, wyniki są słabe. Dla 1 architektury nie udało mi się stworzyć modelu który by nie robił underfittingu — wszystkie mniej czy więcej to robiły, a dla zachowania podobnych eksperymentów co Kuba to nie zmieniałam learning rate. Zmieniałam relu, conv layers, filters ale obecny model to mój najlepszy. Jeśli chodzi o Archi 2 to też mój najlepszy model który stworzyłam, mamy znowu lekki underfitting wiadomo że dla 2 cech będzie to znacznie gorsze wyniki w szczególności dla tak skomplikowanych i nieczytelnych obrazków jak CIFAR. – Model Kuby dla Cifara jest better bo... — tutaj wyjaśnienie czemu jest taki boski.

### Wpływ augmentacji danych