PODSTAWY SIECI NEURONOWYCH

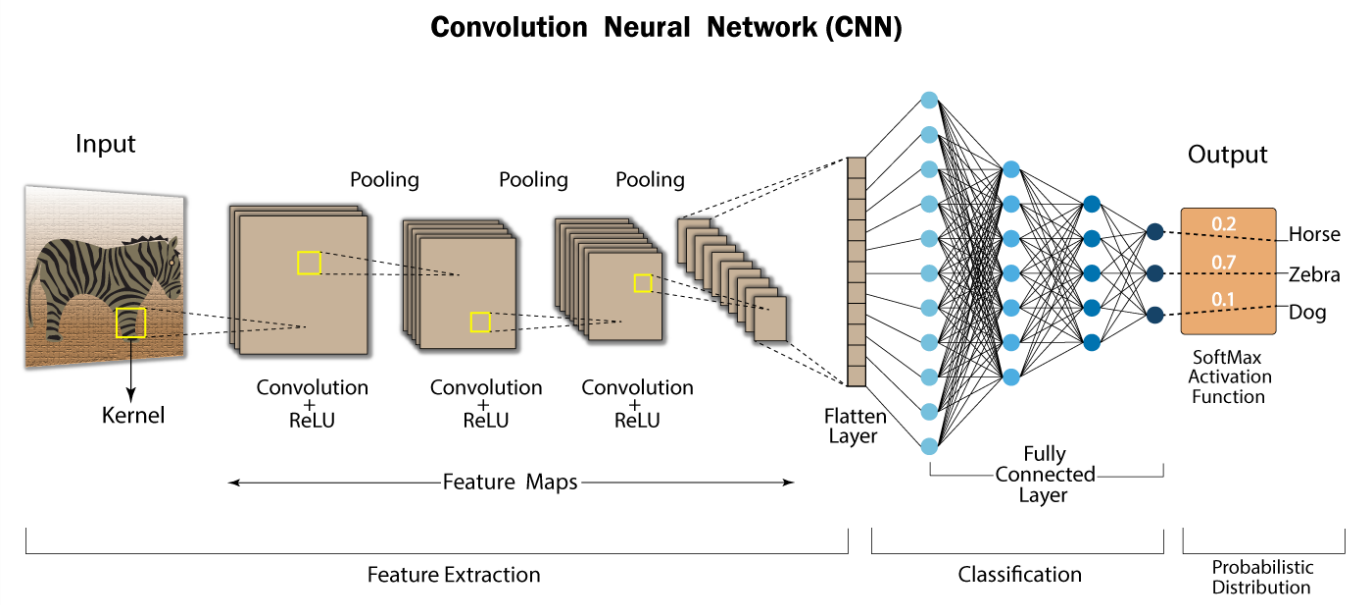
Tobiasz Rolla Piotr Ges

ZDEFINIOWANIE PROBLEMU

- Klasyfikacja kart do gry na podstawie obrazów przy użyciu sieci konwolucyjnej(CNN)
- Klasyfikacja odbywa się w obrębie 53 klas (13 kart z każdego koloru + Jocker)

OPIS DZIAŁANIA SIECI KONWOLUCYJNEJ

- Sieć CNN składa się z warstw
- konwolucyjnej
- poolingowej
- Warstwa w pełni połączona



WARSTWA KONWOLUCYJNA

Celem tej warstwy jest wyodrębnienie szczegółów obrazu poprzez operacji konwolucji(splotu) operacje te wykonuje się poprzez przesuwanie jądra(kernela).

- Padding pading jest techniką pozwalającą zachować rozmiar obrazy

WARSTWA POOLINGOWA

Warstwa ta redukuje rozmiar obrazu przy zachowaniu interesujących szczegłów jej rodzaje to.

- MaxPool największa wartość z kernela zostaje zachowana
- MinPool najmniejsza wartość z kernela zostaje zachowana
- AvragePool liczy średnią wartość dla kernela

WARSTWA W PEŁNI POŁĄCZONA

Warstwa w której każdy z neuronów warstwy jest połączony z każdym neuronem warstwy następnej w sieci.

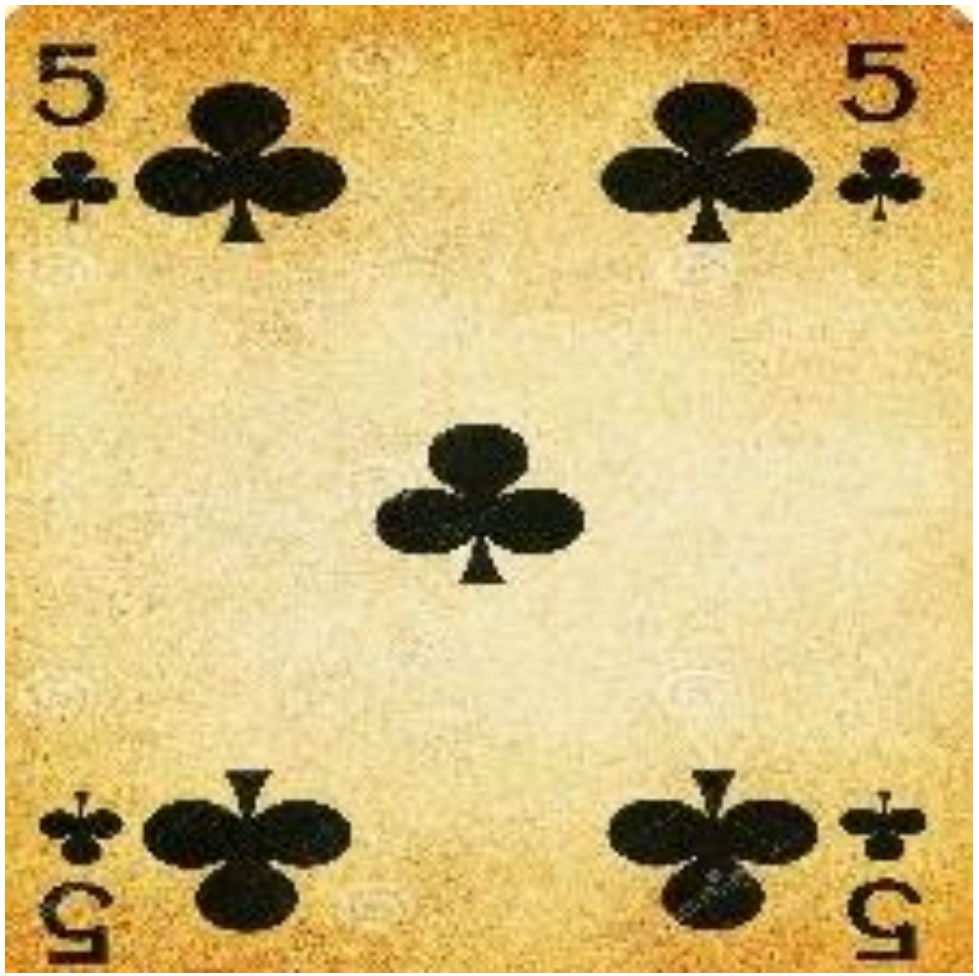
- Bezpośrednio odpowiada za identyfikacje
- Popularne funkcje aktywacji to ReLU i SoftMax(na wyjściu)

ZESTWA DANYCH

Zestaw danych składa się z 8154 kart zestaw został podzielony na 3 zestawy

- zestaw uczący 7624 kart odpowiadając 93.5%
- walidacyjny zawierał 265 kart odpowiadając 3.25%.
- testowy zawierał również 265 kart odpowiadając 3.25%
- Przeprowadzono również testy innych rozkładów zestawu


A
A


5
5
5
5

PRZYGOTOWANIE OBRAZÓW

- Obrazy przekonwertowane do datasetów
- Załadowane za pomocą dataloderów
- Przyjęty dla większości testów bach_size = 32

UCZENIE MODELU

- Funkcja ucząca dostosowywuje parametry modelu.
- Aktualizacja wag za pomocą wstecznej propagacji gradientów.
- Dla każdej pozycji train_loder obliczna jest strata.

```
epoch = 0
done = False
while epoch < num_epochs and not done:
    epoch += 1
    # Training phase
    model.train()
    running_loss, running_corrects = 0.0, 0
    for images, labels in tqdm(train_loader, desc='Training loop'):
        # Move inputs and labels to the device
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * labels.size(0)
        _, preds = torch.max(outputs, 1)  # Pobieranie prognoz
        running_corrects += (preds == labels).sum().item()
    train_loss = running_loss / len(train_loader.dataset)
    train_accuracy = running_corrects / len(train_loader.dataset)
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)
```

FUNKCJA EWAULUJĄCA

Jej celem jest sprawdzenie działania sieci na zbiorze walidacyjnym

- Liczy stratę dla zbioru walidacyjnego

```
# Validation phase
model.eval()
running_loss, running_corrects = 0.0, 0
with torch.no_grad():
    for images, labels in tqdm(val_loader, desc='Validation loop'):
        # Move inputs and labels to the device
        images, labels = images.to(device), labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)
        running_loss += loss.item() * labels.size(0)
        _, preds = torch.max(outputs, 1)  # Pobieranie prognoz
        running_corrects += (preds == labels).sum().item()
val_loss = running_loss / len(val_loader.dataset)
val_accuracy = running_corrects / len(val_loader.dataset)
val_losses.append(val_loss)
val_accuracies.append(val_accuracy)
print(f"Epoch {epoch}/{num_epochs} - Train loss: {train_loss:.4f}, Validation loss: {val_loss:.4f}, Train acc: {train_accuracy:.4f}, Val acc: {val_accuracy:.4f}")

# Aktualizacja harmonogramu
#scheduler.step(val_loss)
 # Check early stopping criteria
done = es(model, val_loss)
print(f"Early Stopping: {es.status}")
```

WCZESNE ZATRZYMANIE

W celu zapobiegnięciu przeuczeniu zastosowano wsteczne zatrzymanie.

- Jeśli dokładność nie wzrośnie przez n epok
- Model wróci do epoki gdzie nie zauważono spadku strat dokładności

```
class EarlyStopping:
    def __init__(self, patience=5, min_delta=0, restore_best_weights=True):
        self.patience = patience
        self.min_delta = min_delta
        self.restore_best_weights = restore_best_weights
        self.best_model = None
        self.best_loss = float('inf')
        self.counter = 0
        self.status = ""

    def __call__(self, model, val_loss):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.counter = 0
            self.best_model = copy.deepcopy(model.state_dict())
        else:
            self.counter += 1
            if self.counter >= self.patience:
                self.status = "Stopping training"
                if self.restore_best_weights:
                    model.load_state_dict(self.best_model)
                return True
        self.status = f"{self.counter}/{self.patience}"
        return False
```

SPRAWDZENIE MODELU

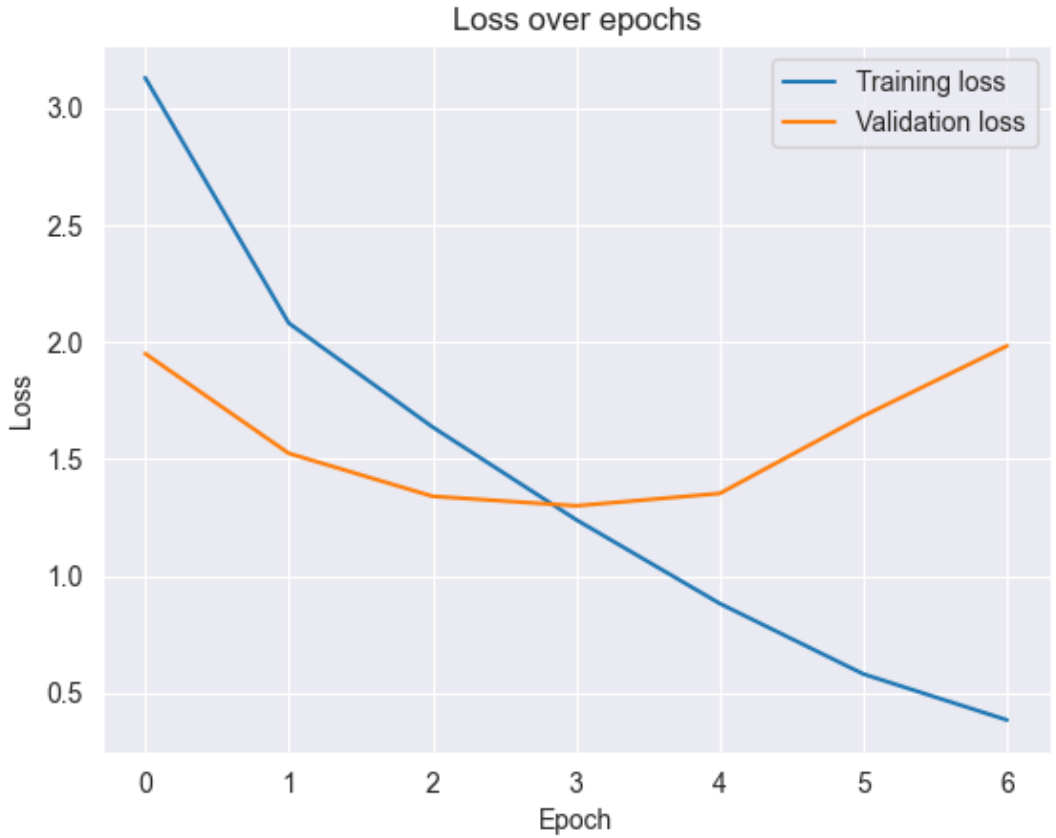
W celu sprawdzenia modelu po zakończonym procesie nauki liczy się dokładność na zbiorze testowym.

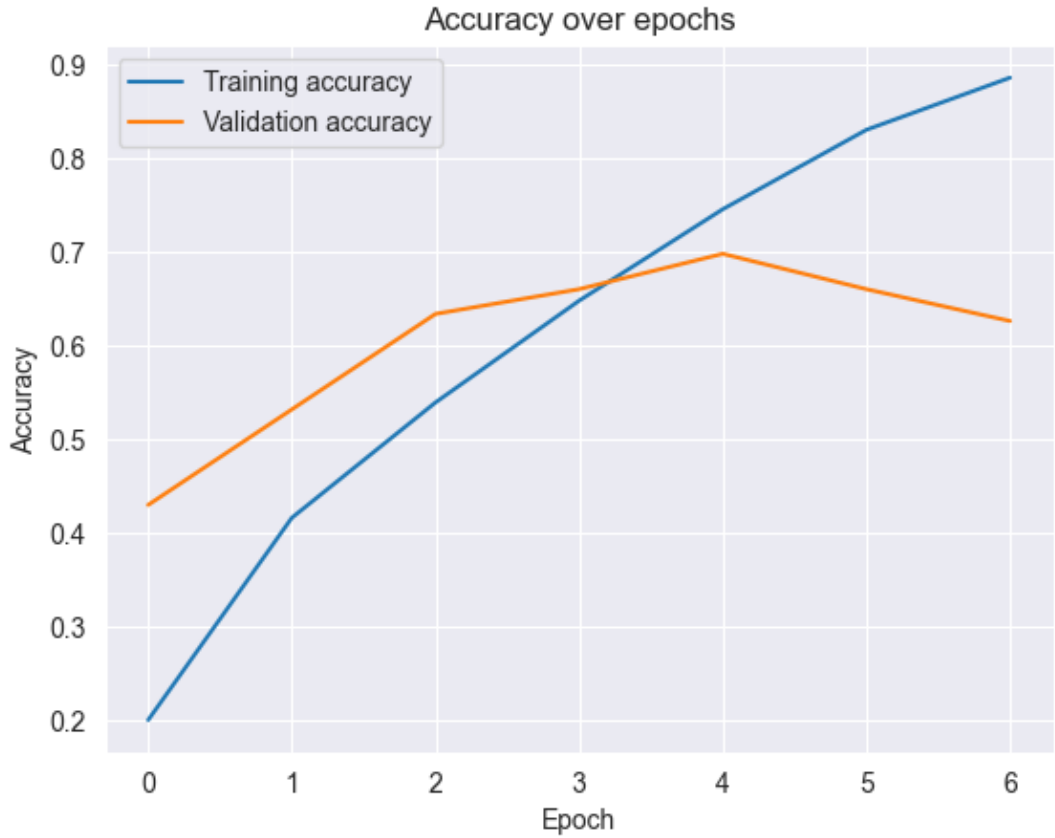
```
model.eval()
true_labels = []
predicted_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        outputs = model(images)
        _, predictions = torch.max(outputs, 1)  # Get the class index with the highest probability
        true_labels.extend(labels.cpu().numpy())
        predicted_labels.extend(predictions.cpu().numpy())

# Calculate accuracy
accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

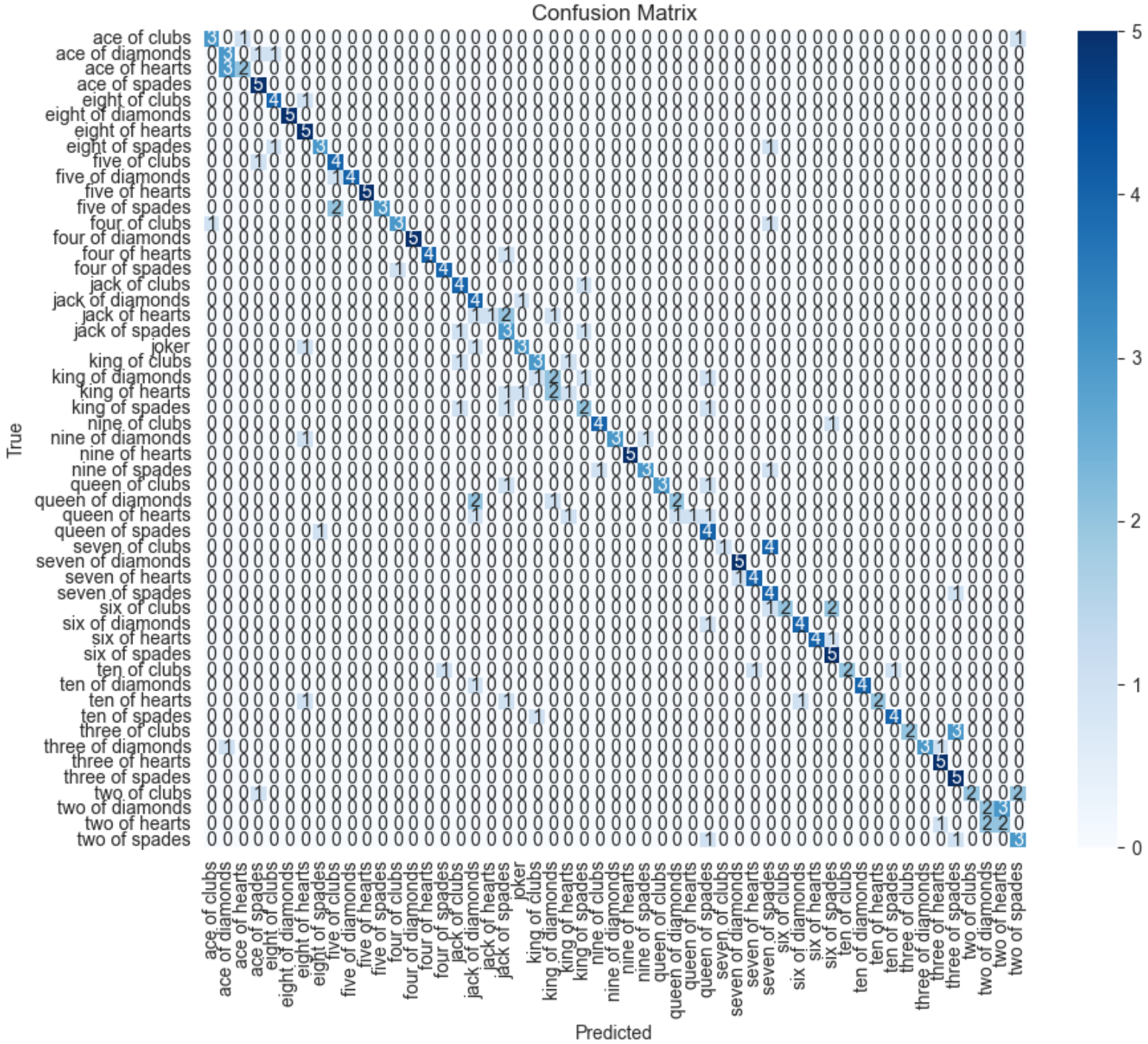
BADANIE 1

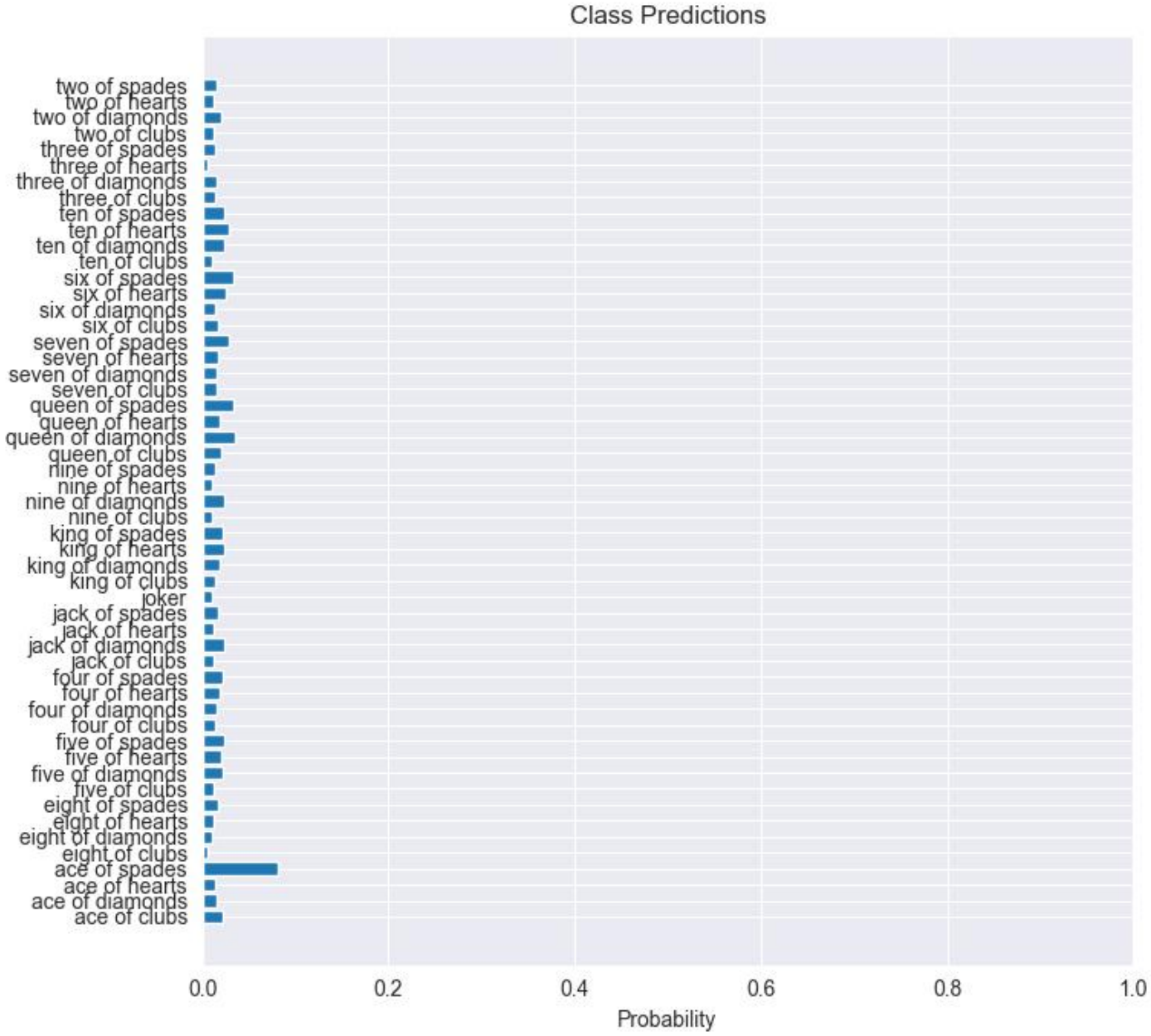
Badanie na sieci składającej się z 6 warstw 2 warstwy kowolucyjen 2 poolingowe 2 w pełni połączone

```python
#Model
class CardClassifier(nn.Module):
    def __init__(self, num_classes=53):
        super(CardClassifier,self).__init__()

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(128 * 32 * 32, 512)
        self.fc2 = nn.Linear(512, num_classes)



    def forward(self, x):

        x = self.pool1(self.relu1((self.conv1(x))))
        x = self.pool2(self.relu2((self.conv2(x))))
        x = x.view(-1, 128*32*32)
        x = F.relu(self.fc1(x))
        output = self.fc2(x)

        return output
```


Loss over epochs
Training loss
Validation loss
Loss
Epoch
3.0
2.5
2.0
1.5
1.0
0.5
0
1
2
3
4
5
6


Accuracy over epochs
Training accuracy
Validation accuracy
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
1
2
3
4
5
6


Confusion Matrix
True
Predicted
ace of clubs
ace of diamonds
ace of hearts
ace of spades
eight of clubs
eight of diamonds
eight of hearts
eight of spades
five of clubs
five of diamonds
five of hearts
five of spades
four of clubs
four of diamonds
four of hearts
four of spades
jack of clubs
jack of diamonds
jack of hearts
jack of spades
joker
king of clubs
king of diamonds
king of hearts
king of spades
nine of clubs
nine of diamonds
nine of hearts
nine of spades
queen of clubs
queen of diamonds
queen of hearts
queen of spades
seven of clubs
seven of diamonds
seven of hearts
seven of spades
six of clubs
six of diamonds
six of hearts
six of spades
ten of clubs
ten of diamonds
ten of hearts
ten of spades
three of clubs
three of diamonds
three of hearts
three of spades
two of clubs
two of diamonds
two of hearts
two of spades
5
4
3
2
1
0


Class Predictions
two of spades
two of hearts
two of diamonds
two of clubs
three of spades
three of hearts
three of diamonds
three of clubs
ten of spades
ten of hearts
ten of diamonds
ten of clubs
six of spades
six of hearts
six of diamonds
six of clubs
seven of spades
seven of hearts
seven of diamonds
seven of clubs
queen of spades
queen of hearts
queen of diamonds
queen of clubs
nine of spades
nine of hearts
nine of diamonds
nine of clubs
king of spades
king of hearts
king of diamonds
king of clubs
joker
jack of spades
jack of hearts
jack of diamonds
jack of clubs
four of spades
four of hearts
four of diamonds
four of clubs
five of spades
five of hearts
five of diamonds
five of clubs
eight of spades
eight of hearts
eight of diamonds
eight of clubs
ace of spades
ace of hearts
ace of diamonds
ace of clubs
0.0
0.2
0.4
0.6
0.8
1.0
Probability

WNIOSKI DO BADANIA 1

Sieć uczyła się bardzo szybko nie będąc odporną na przeuczenie.

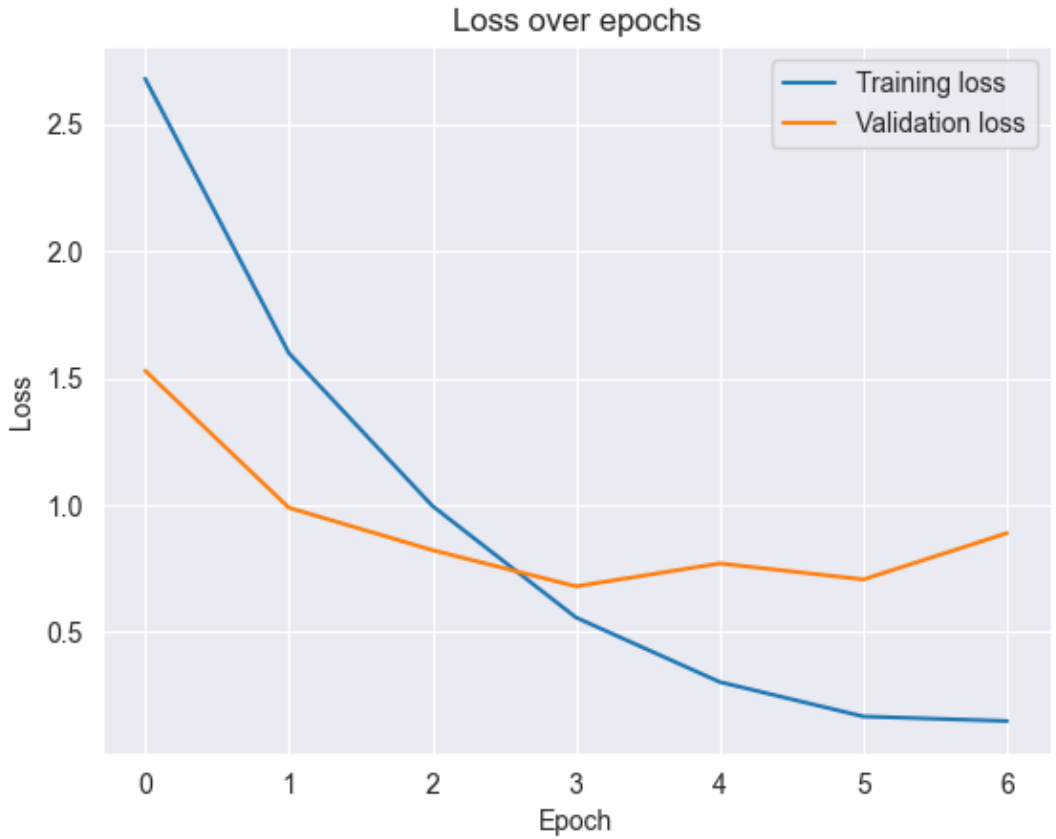
- Dokładność 66.04%
- Po 4 epoce nastąpiło przeuczanie

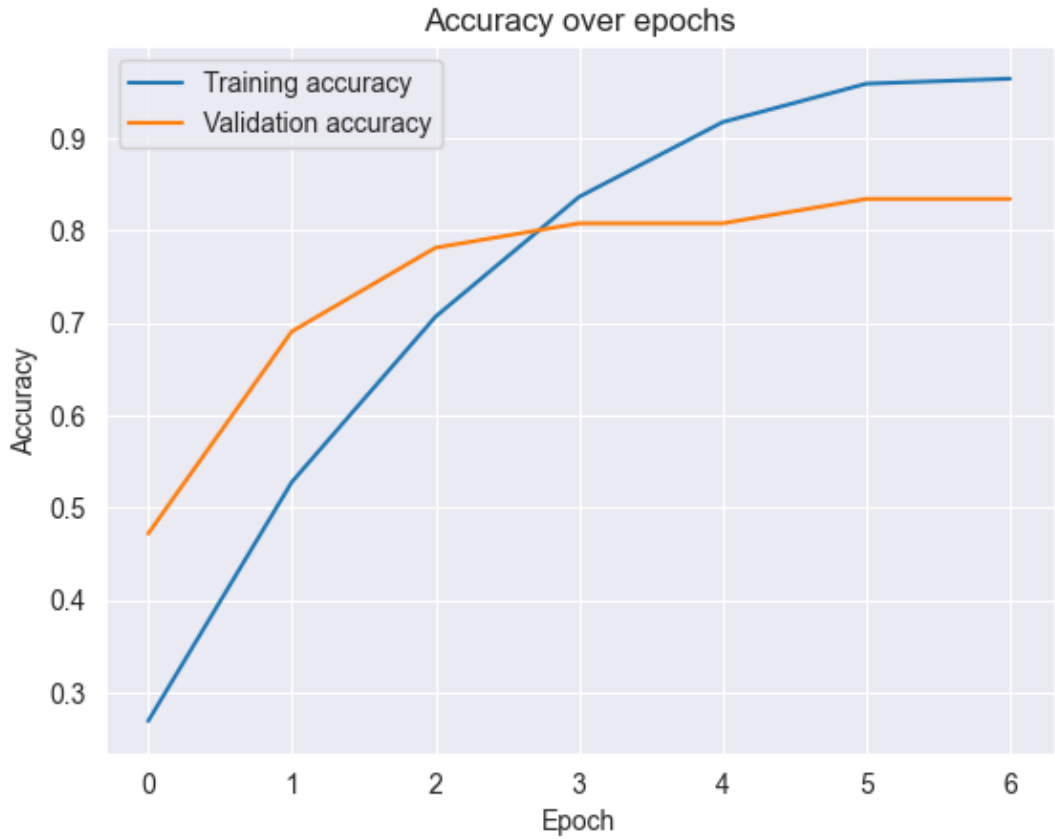
BADANIE 2

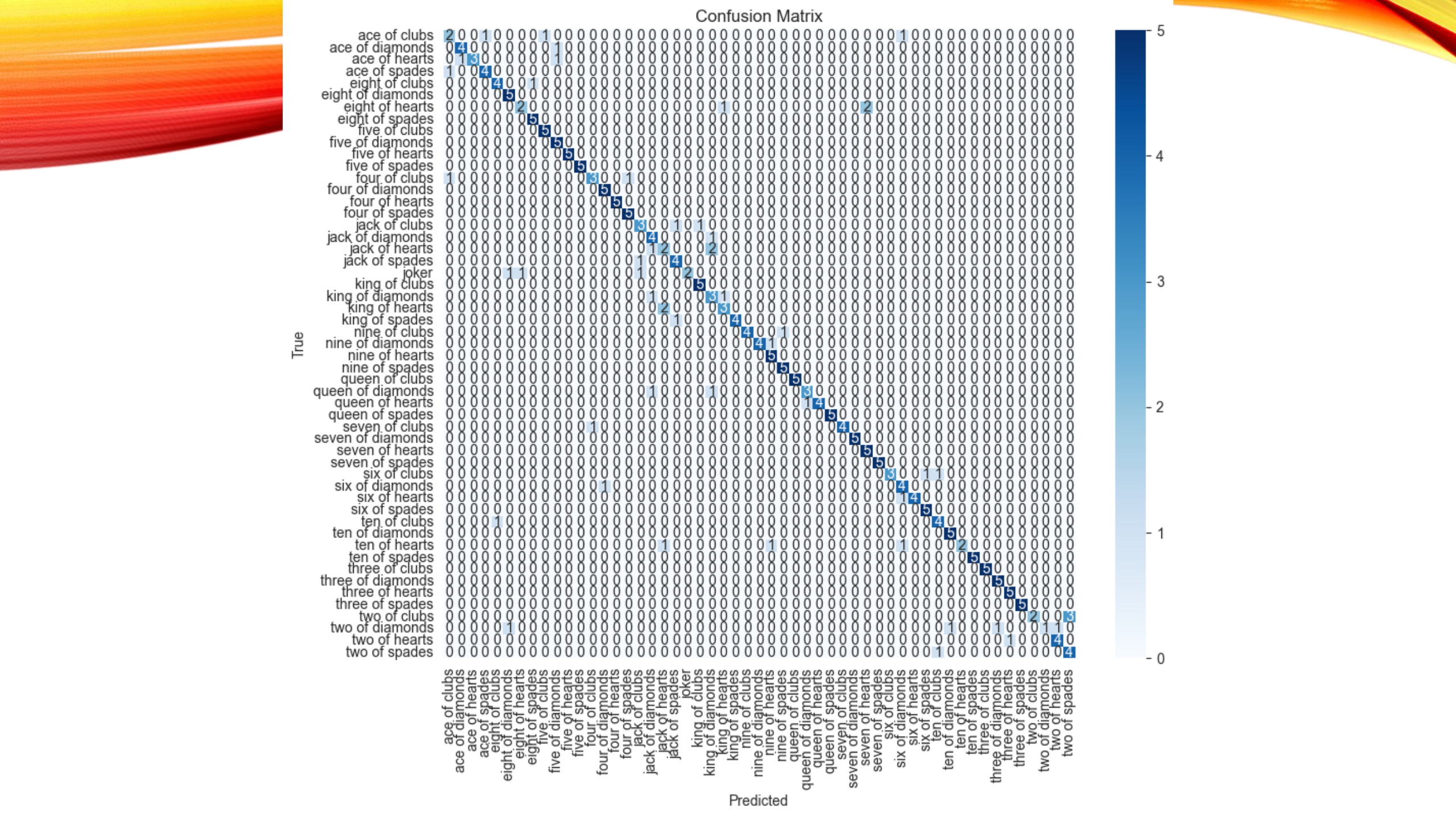
W badaniu drugim dodano 3 warstwy

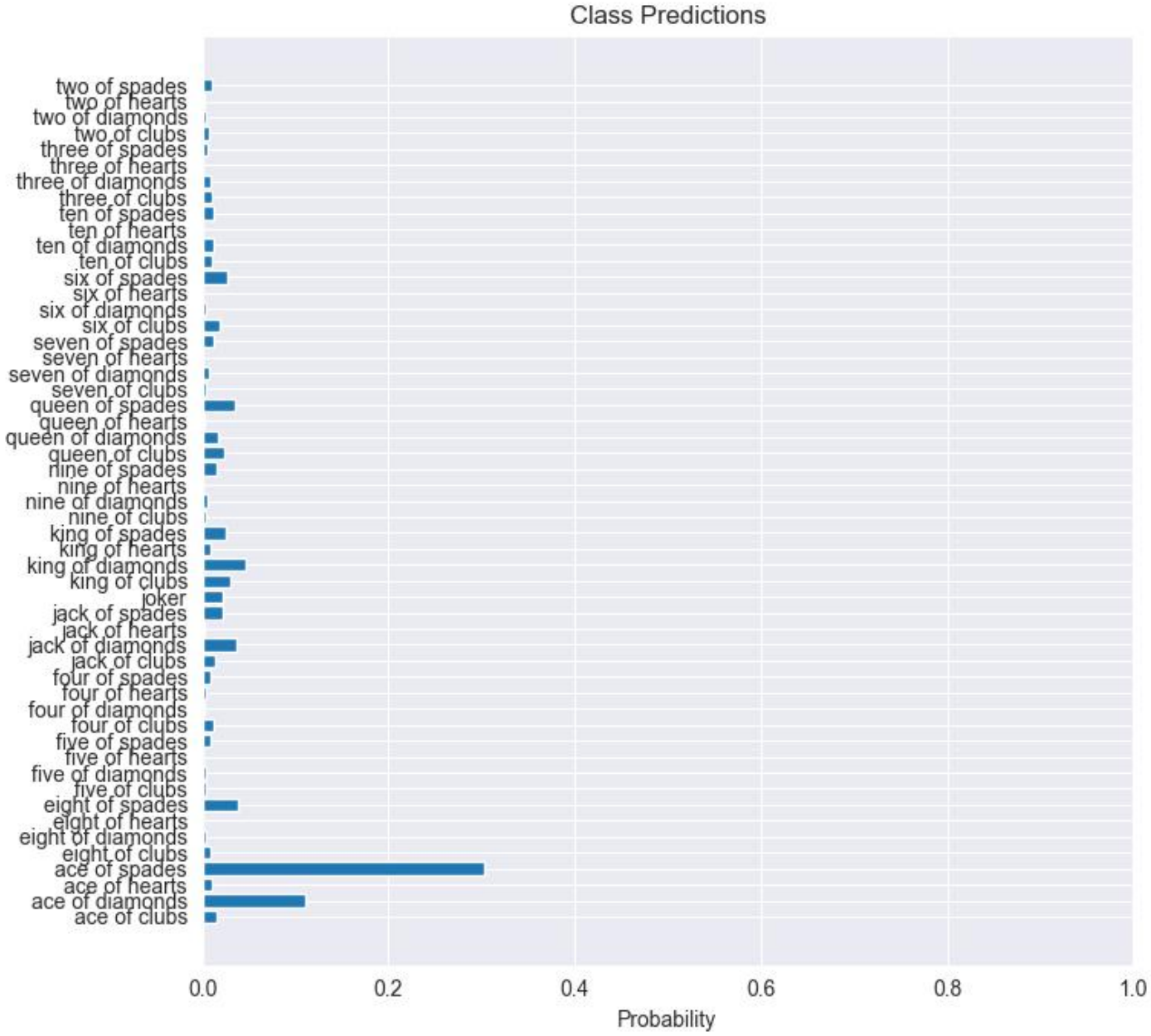
- 1 warstwa konolucyjna
- 1 warstwa poolingowa
- 1 warstwa jednolicie połączona

```python
#Model
class CardClassifier(nn.Module):
    def __init__(self, num_classes=53):
        super(CardClassifier,self).__init__()

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(128 * 16 * 16, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool(F.relu((self.conv1(x))))
        x = self.pool(F.relu((self.conv2(x))))
        x = self.pool(F.relu((self.conv3(x))))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        output = self.fc3(x)

        return output
```


Loss over epochs
Training loss
Validation loss
Loss
Epoch
2.5
2.0
1.5
1.0
0.5
0
1
2
3
4
5
6


Accuracy over epochs
Training accuracy
Validation accuracy
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
1
2
3
4
5
6


Confusion Matrix
True
Predicted
ace of clubs
ace of diamonds
ace of hearts
ace of spades
eight of clubs
eight of diamonds
eight of hearts
eight of spades
five of clubs
five of diamonds
five of hearts
five of spades
four of clubs
four of diamonds
four of hearts
four of spades
jack of clubs
jack of diamonds
jack of hearts
jack of spades
joker
king of clubs
king of diamonds
king of hearts
king of spades
nine of clubs
nine of diamonds
nine of hearts
nine of spades
queen of clubs
queen of diamonds
queen of hearts
queen of spades
seven of clubs
seven of diamonds
seven of hearts
seven of spades
six of clubs
six of diamonds
six of hearts
six of spades
ten of clubs
ten of diamonds
ten of hearts
ten of spades
three of clubs
three of diamonds
three of hearts
three of spades
two of clubs
two of diamonds
two of hearts
two of spades
0
1
2
3
4
5


Class Predictions
two of spades
two of hearts
two of diamonds
two of clubs
three of spades
three of hearts
three of diamonds
three of clubs
ten of spades
ten of hearts
ten of diamonds
ten of clubs
six of spades
six of hearts
six of diamonds
six of clubs
seven of spades
seven of hearts
seven of diamonds
seven of clubs
queen of spades
queen of hearts
queen of diamonds
queen of clubs
nine of spades
nine of hearts
nine of diamonds
nine of clubs
king of spades
king of hearts
king of diamonds
king of clubs
joker
jack of spades
jack of hearts
jack of diamonds
jack of clubs
four of spades
four of hearts
four of diamonds
four of clubs
five of spades
five of hearts
five of diamonds
five of clubs
eight of spades
eight of hearts
eight of diamonds
eight of clubs
ace of spades
ace of hearts
ace of diamonds
ace of clubs
0.0
0.2
0.4
0.6
0.8
1.0
Probability

WNIOSKI DO BADANIA 2

Dodatkowe warstwy poprawiły dokładność nie zapobiegły przeuczaniu

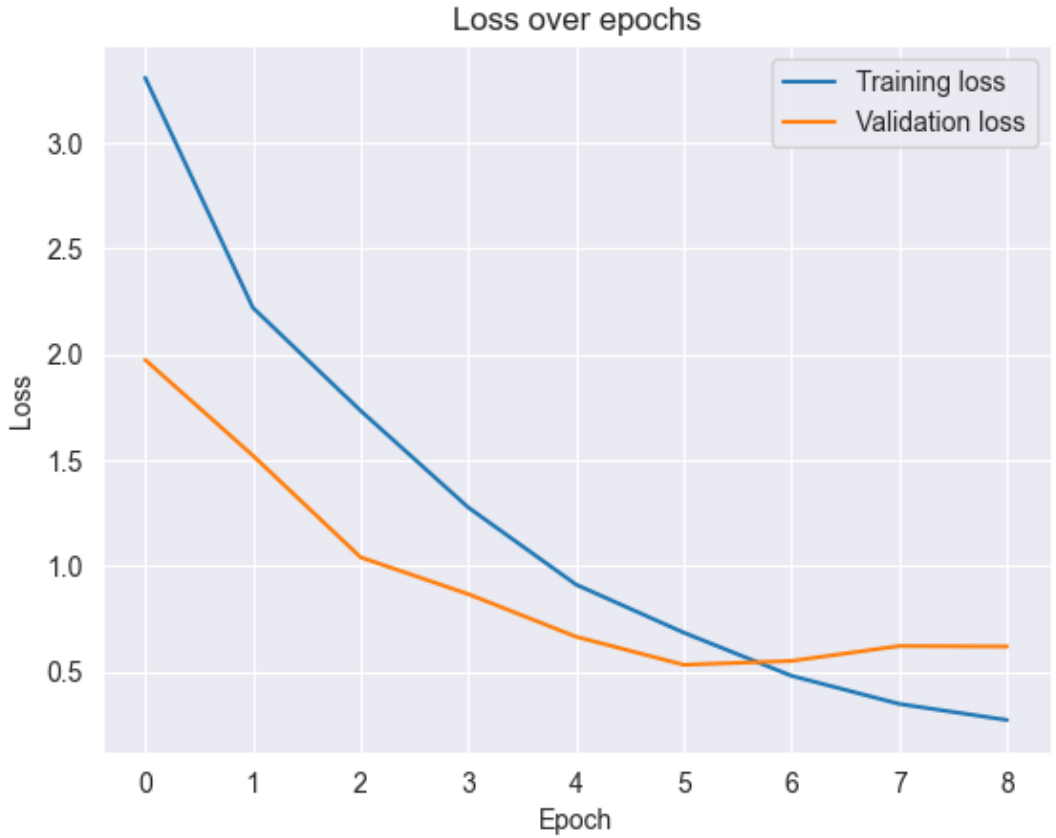
- Dokładność 80.75%
- Ilość epok przed przeuczeniem 4

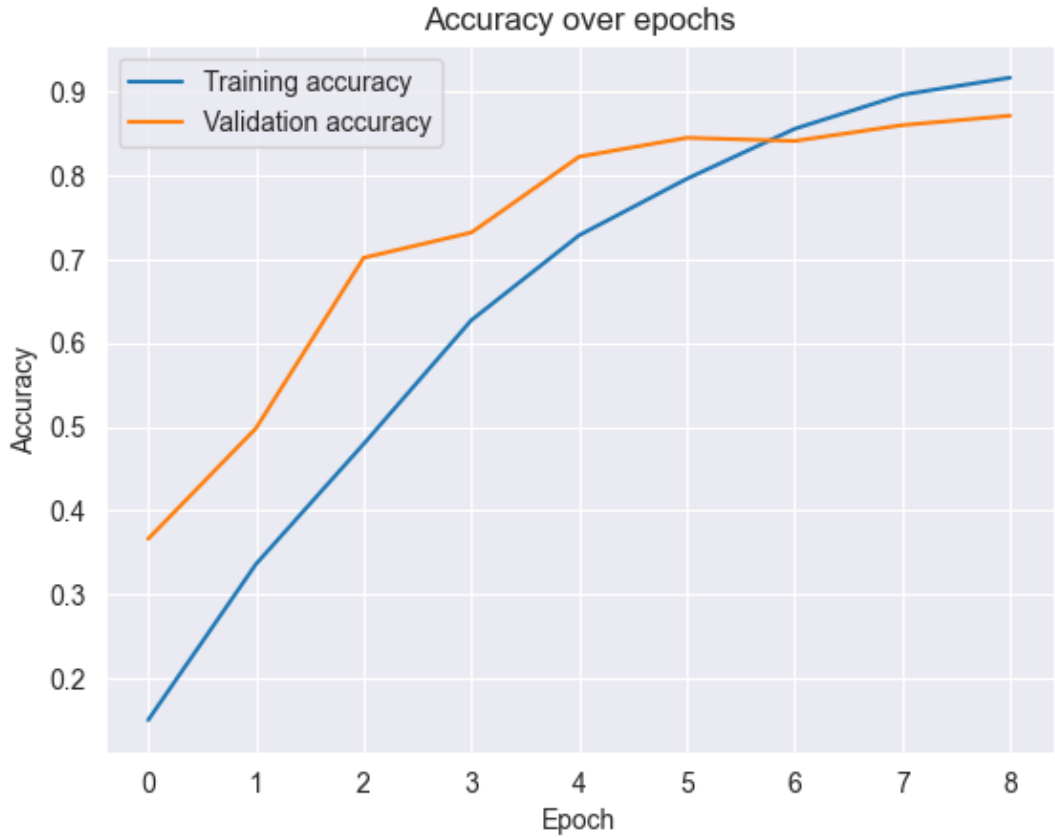
BADANIE 3

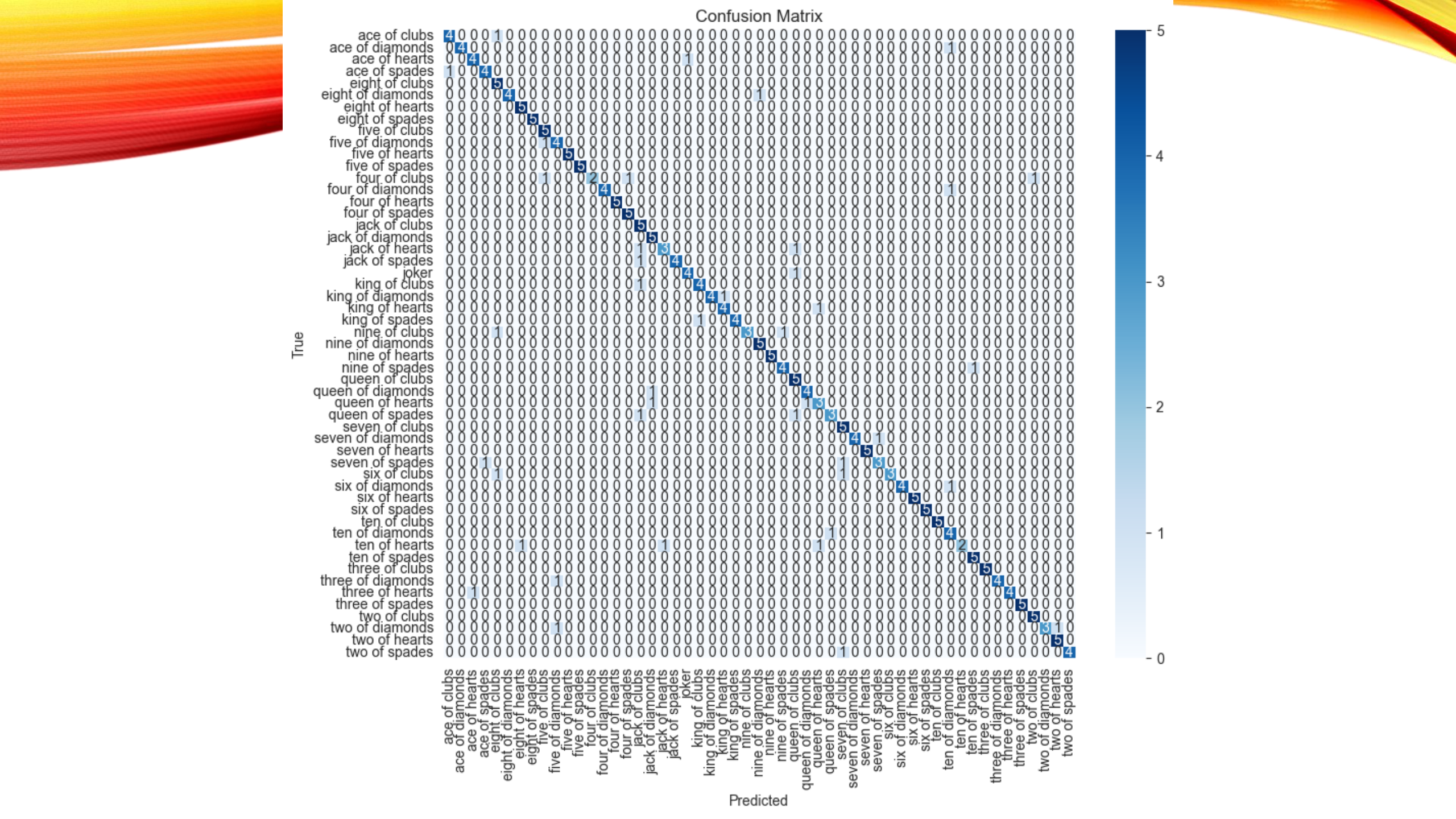
W 3 modelu dodano:

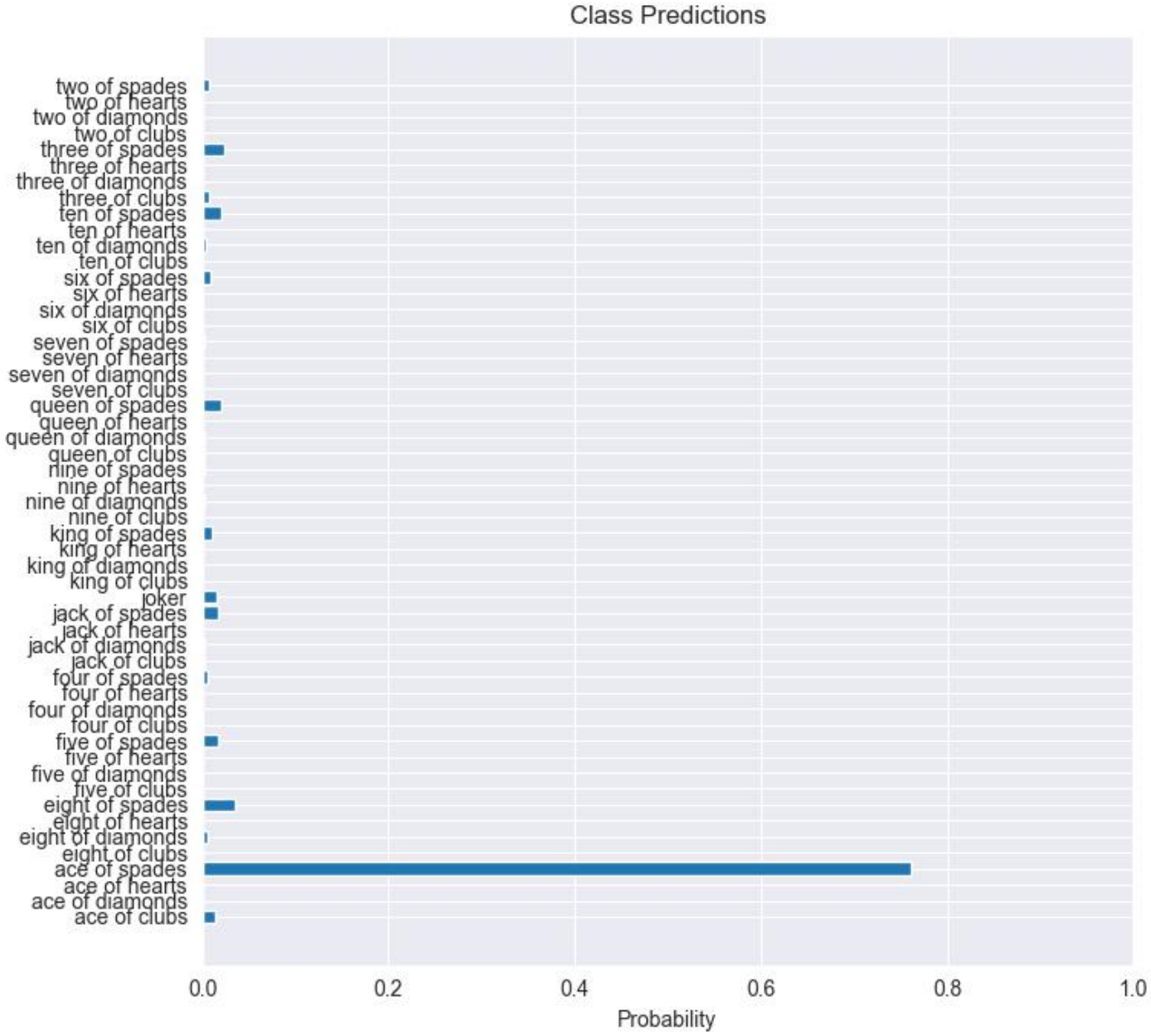
- Bach Normalisation po każdej warstwie konwolucyjnej
- Zmodyfikowano funkcje Adam o weight-decade

```python
#Model
class CardClassifier(nn.Module):
    def __init__(self, num_classes=53):
        super(CardClassifier,self).__init__()

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(128 * 16 * 16, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        output = self.fc3(x)

        return output
```


Loss over epochs
Training loss
Validation loss
Loss
Epoch
3.0
2.5
2.0
1.5
1.0
0.5
0
1
2
3
4
5
6
7
8


Accuracy over epochs
Training accuracy
Validation accuracy
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
1
2
3
4
5
6
7
8


Confusion Matrix
True
Predicted
ace of clubs
ace of diamonds
ace of hearts
ace of spades
eight of clubs
eight of diamonds
eight of hearts
eight of spades
five of clubs
five of diamonds
five of hearts
five of spades
four of clubs
four of diamonds
four of hearts
four of spades
jack of clubs
jack of diamonds
jack of hearts
jack of spades
joker
king of clubs
king of diamonds
king of hearts
king of spades
nine of clubs
nine of diamonds
nine of hearts
nine of spades
queen of clubs
queen of diamonds
queen of hearts
queen of spades
seven of clubs
seven of diamonds
seven of hearts
seven of spades
six of clubs
six of diamonds
six of hearts
six of spades
ten of clubs
ten of diamonds
ten of hearts
ten of spades
three of clubs
three of diamonds
three of hearts
three of spades
two of clubs
two of diamonds
two of hearts
two of spades
5
4
3
2
1
0


Class Predictions
two of spades
two of hearts
two of diamonds
two of clubs
three of spades
three of hearts
three of diamonds
three of clubs
ten of spades
ten of hearts
ten of diamonds
ten of clubs
six of spades
six of hearts
six of diamonds
six of clubs
seven of spades
seven of hearts
seven of diamonds
seven of clubs
queen of spades
queen of hearts
queen of diamonds
queen of clubs
nine of spades
nine of hearts
nine of diamonds
nine of clubs
king of spades
king of hearts
king of diamonds
king of clubs
joker
jack of spades
jack of hearts
jack of diamonds
jack of clubs
four of spades
four of hearts
four of diamonds
four of clubs
five of spades
five of hearts
five of diamonds
five of clubs
eight of spades
eight of hearts
eight of diamonds
eight of clubs
ace of spades
ace of hearts
ace of diamonds
ace of clubs
0.0
0.2
0.4
0.6
0.8
1.0
Probability

WNIOSKI DO BADANIA 3

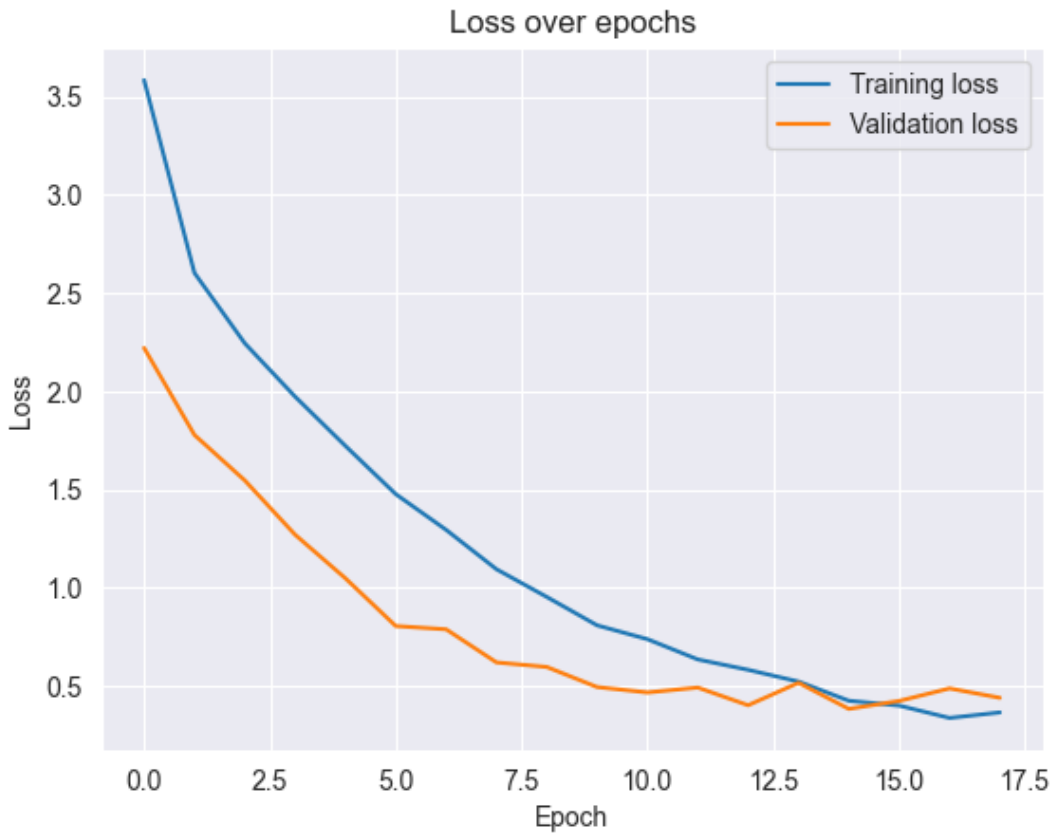
Po wprowadzeniu poprawek sieć jest nadal podatna na przeuczenie jednak w mniejszym stopniu

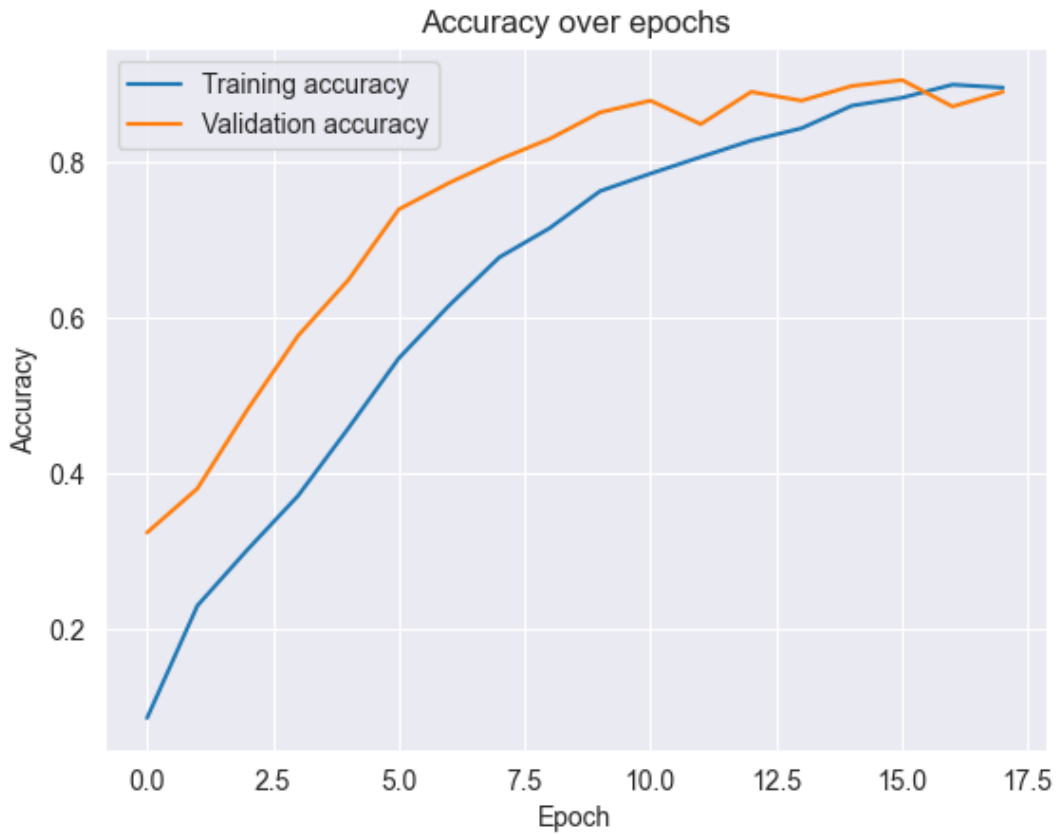
- Dokładność 84.53%
- Ilość epok przed przeuczeniem 6

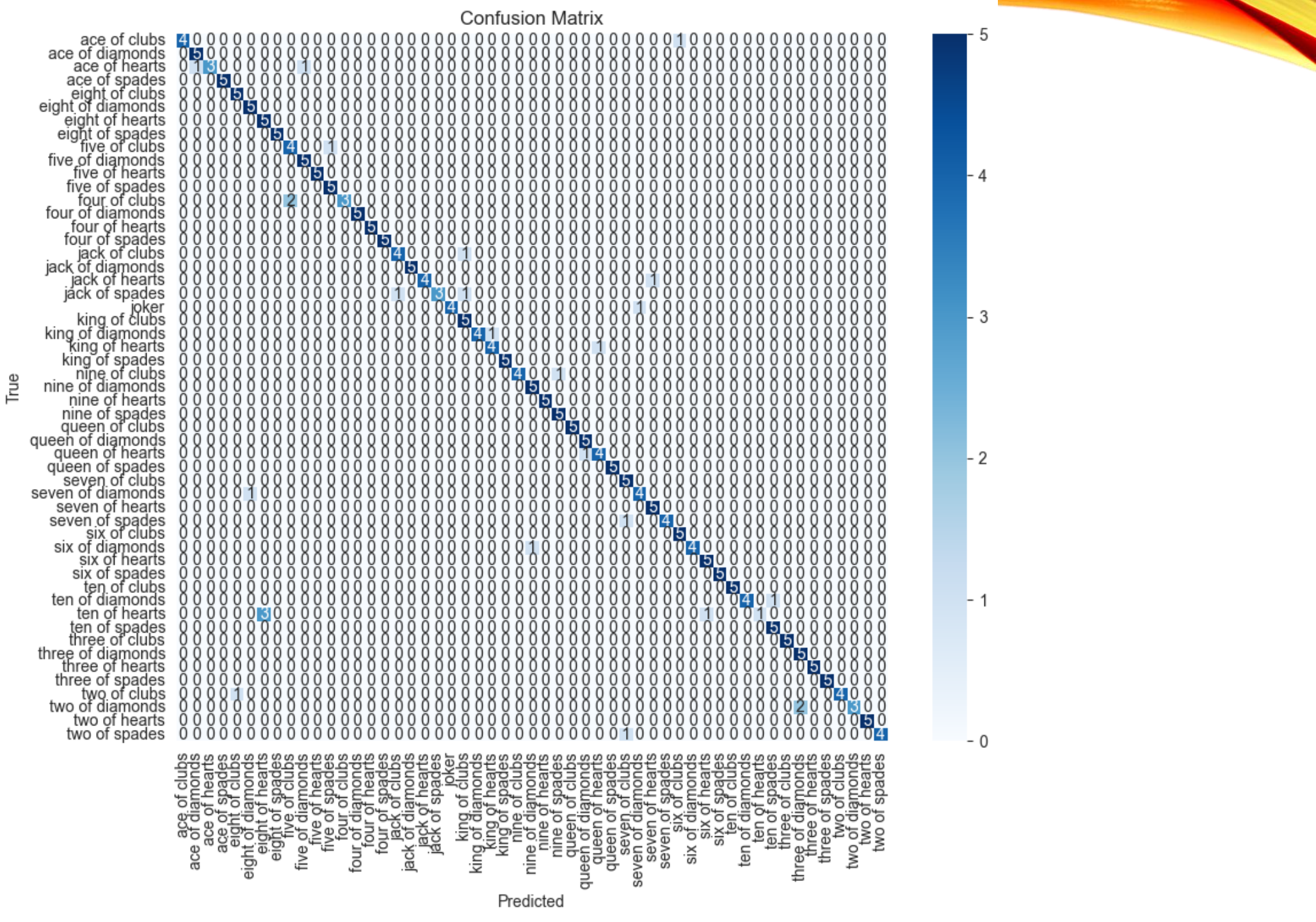
BADANIE 4

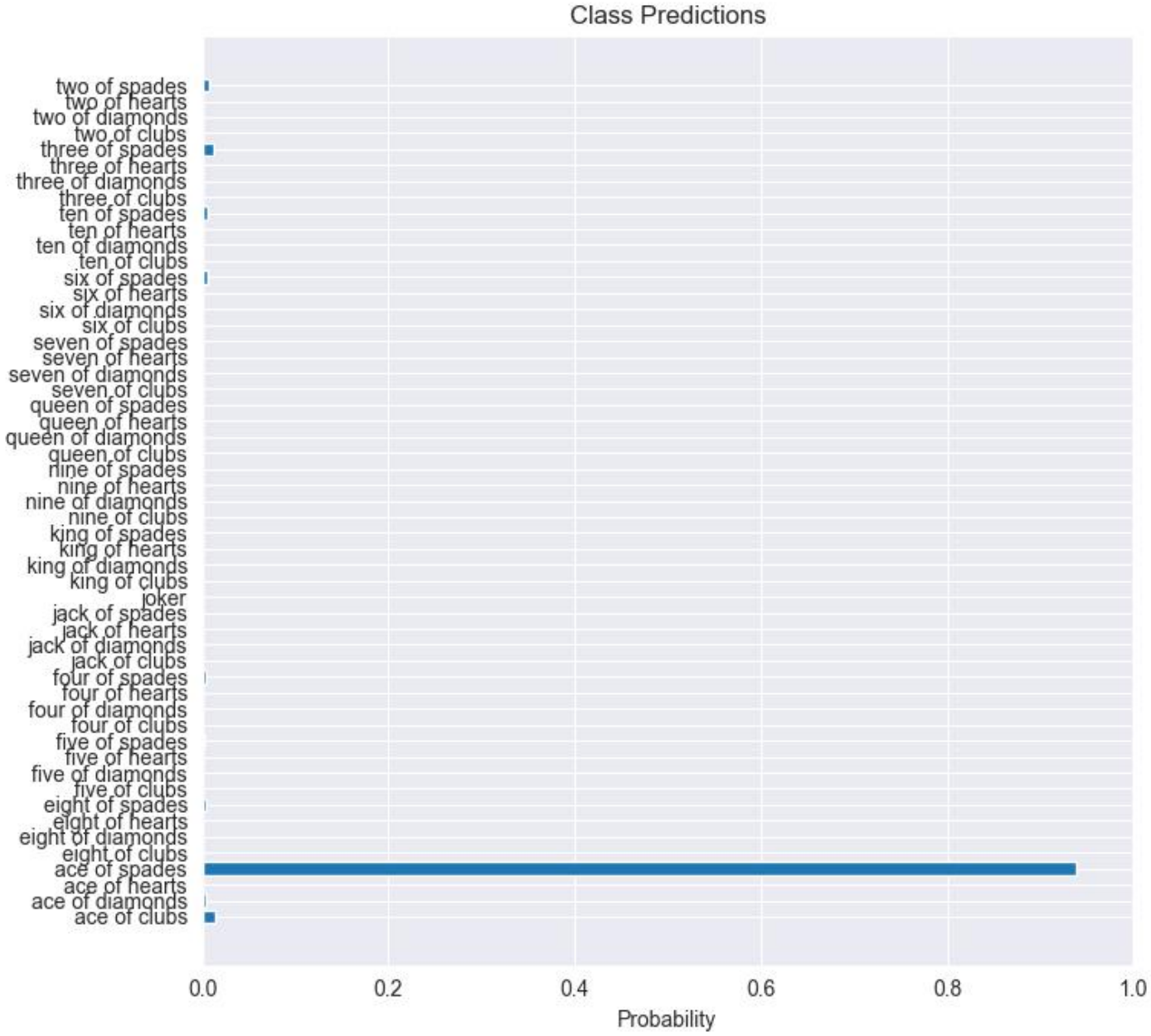
W 4 modelu zastosowano technikę drop-out w celu zapobiegnięciu przeuczeniu.

```python
#Model
class CardClassifier(nn.Module):
    def __init__(self, num_classes=53):
        super(CardClassifier,self).__init__()

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(128 * 16 * 16, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, num_classes)
        self.dropout1 = nn.Dropout(p=0.1)
        self.dropout2 = nn.Dropout(p=0.2)
        self.dropout3 = nn.Dropout(p=0.3)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = self.dropout2(x)
        x = F.relu(self.fc3(x))
        x = self.dropout3(x)
        output = self.fc4(x)

        return output
```


Loss over epochs
Training loss
Validation loss
Loss
Epoch
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Accuracy over epochs
Training accuracy
Validation accuracy
Accuracy
Epoch
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Confusion Matrix
True
Predicted
ace of clubs
ace of diamonds
ace of hearts
ace of spades
eight of clubs
eight of diamonds
eight of hearts
eight of spades
five of clubs
five of diamonds
five of hearts
five of spades
four of clubs
four of diamonds
four of hearts
four of spades
jack of clubs
jack of diamonds
jack of hearts
jack of spades
joker
king of clubs
king of diamonds
king of hearts
king of spades
nine of clubs
nine of diamonds
nine of hearts
nine of spades
queen of clubs
queen of diamonds
queen of hearts
queen of spades
seven of clubs
seven of diamonds
seven of hearts
seven of spades
six of clubs
six of diamonds
six of hearts
six of spades
ten of clubs
ten of diamonds
ten of hearts
ten of spades
three of clubs
three of diamonds
three of hearts
three of spades
two of clubs
two of diamonds
two of hearts
two of spades
0
1
2
3
4
5


Class Predictions
two of spades
two of hearts
two of diamonds
two of clubs
three of spades
three of hearts
three of diamonds
three of clubs
ten of spades
ten of hearts
ten of diamonds
ten of clubs
six of spades
six of hearts
six of diamonds
six of clubs
seven of spades
seven of hearts
seven of diamonds
seven of clubs
queen of spades
queen of hearts
queen of diamonds
queen of clubs
nine of spades
nine of hearts
nine of diamonds
nine of clubs
king of spades
king of hearts
king of diamonds
king of clubs
joker
jack of spades
jack of hearts
jack of diamonds
jack of clubs
four of spades
four of hearts
four of diamonds
four of clubs
five of spades
five of hearts
five of diamonds
five of clubs
eight of spades
eight of hearts
eight of diamonds
eight of clubs
ace of spades
ace of hearts
ace of diamonds
ace of clubs
0.0
0.2
0.4
0.6
0.8
1.0
Probability

WNIOSKI DO BADANIA 4

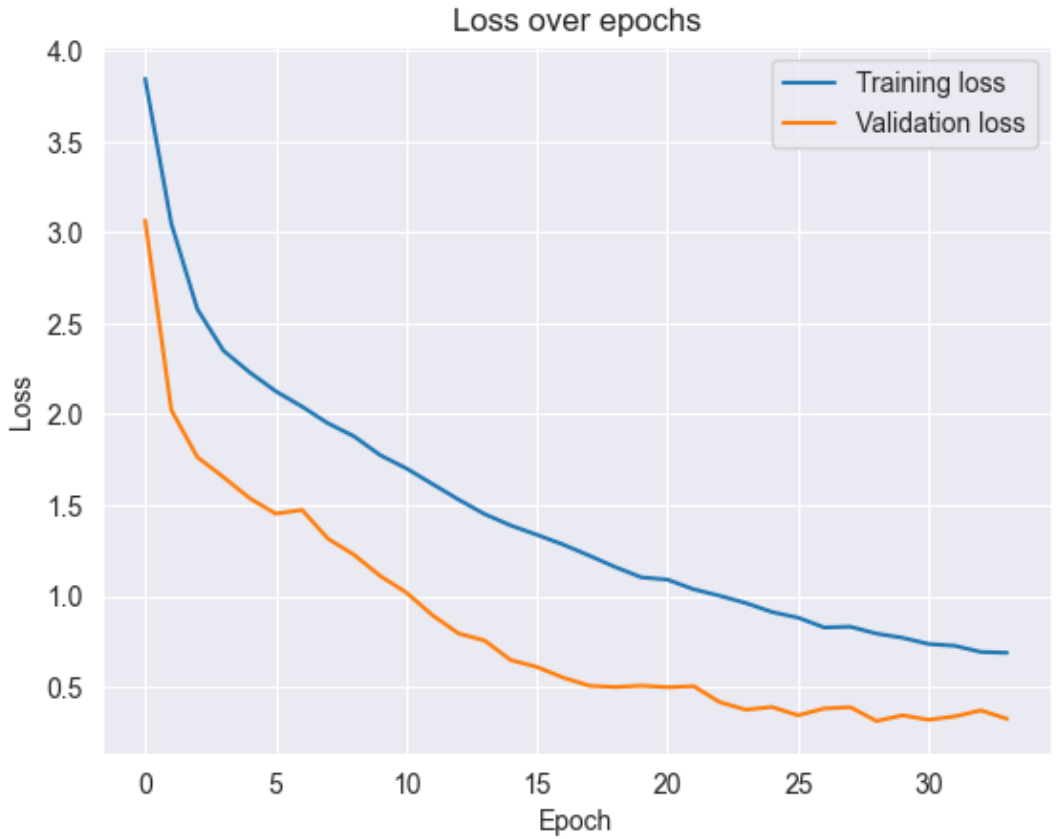
Udało się ograniczyć przeuczenie ponadto wzrosła doładność

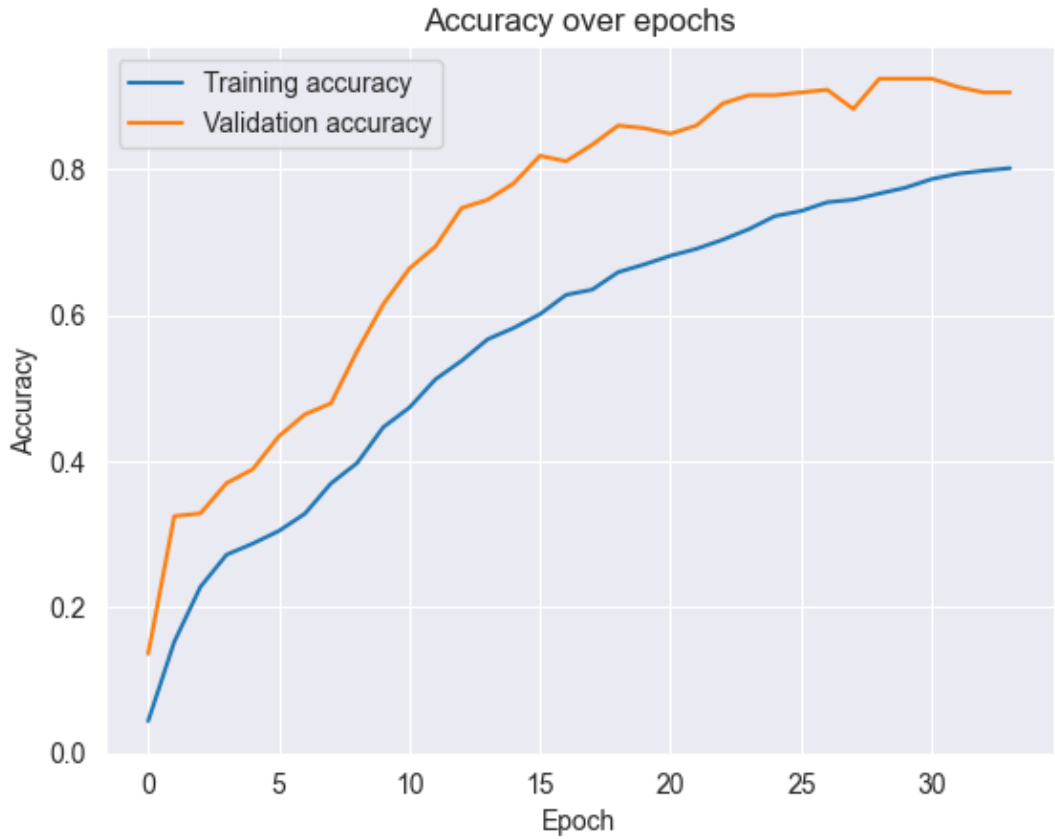
- Dokładność 89.91%
- Liczba epok przed przeuczeniem 16

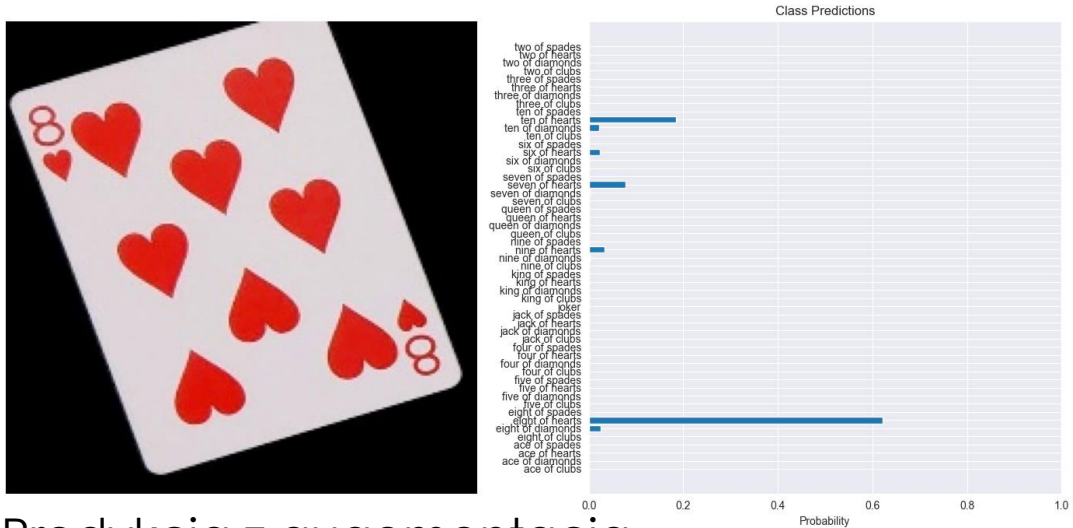
BADANIE 5

W piątym badaniu dodano augumentacje danych.

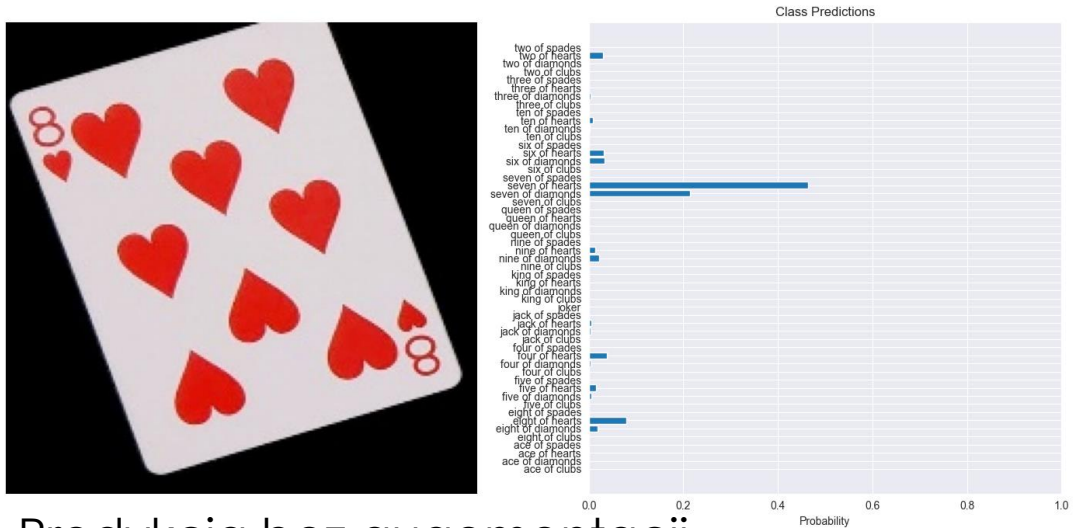
- Losowe obracanie obrazów pionowe poziome
- Losowe obracanie obrazów o 30 stopni
- Losowa zmian kolorów obrazu


Loss over epochs
Training loss
Validation loss
Loss
Epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
30


Accuracy over epochs
Training accuracy
Validation accuracy
Accuracy
Epoch
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30



Predykcja z augementacją



Predykcja bez augementacji

WNIOSKI DO BADANIA 5

Wprowadzenie augumentacji w znaczny sposób zredukowało przeuczenie sieci Ponadto wzrosła dokładność w testach obrazów o niestandardowej strukturze np. obrazy obrócone.

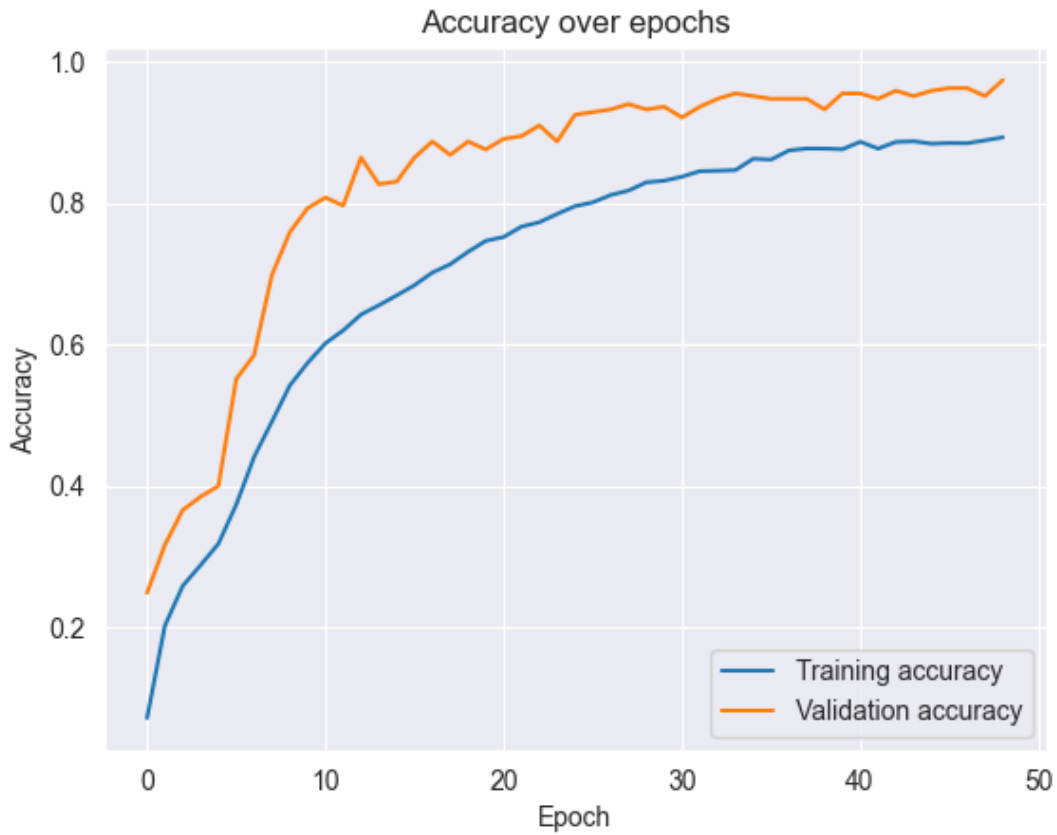
- Dokładność 92.45%
- 27 epok do przeuczenia

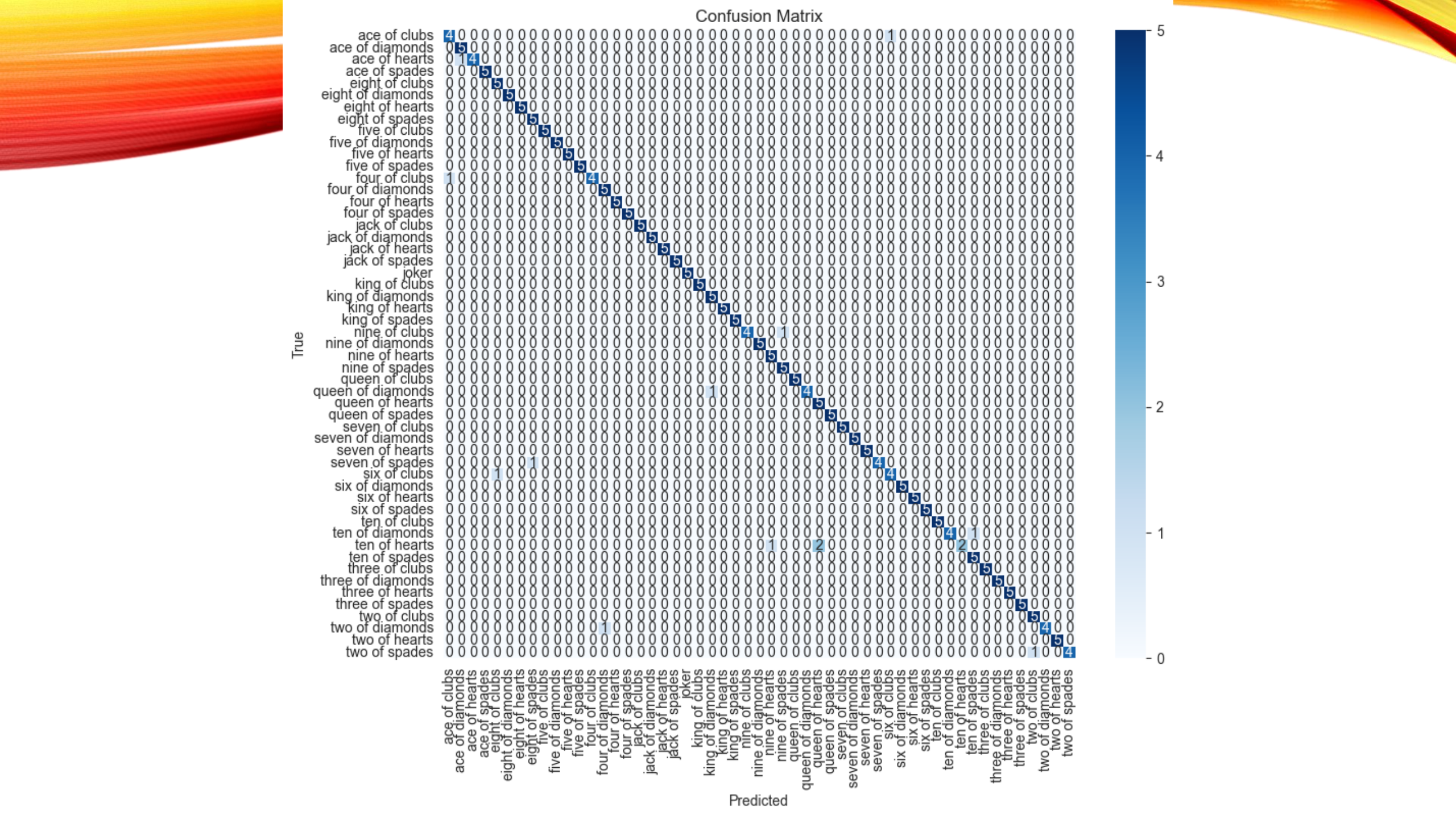
BADANIE 6

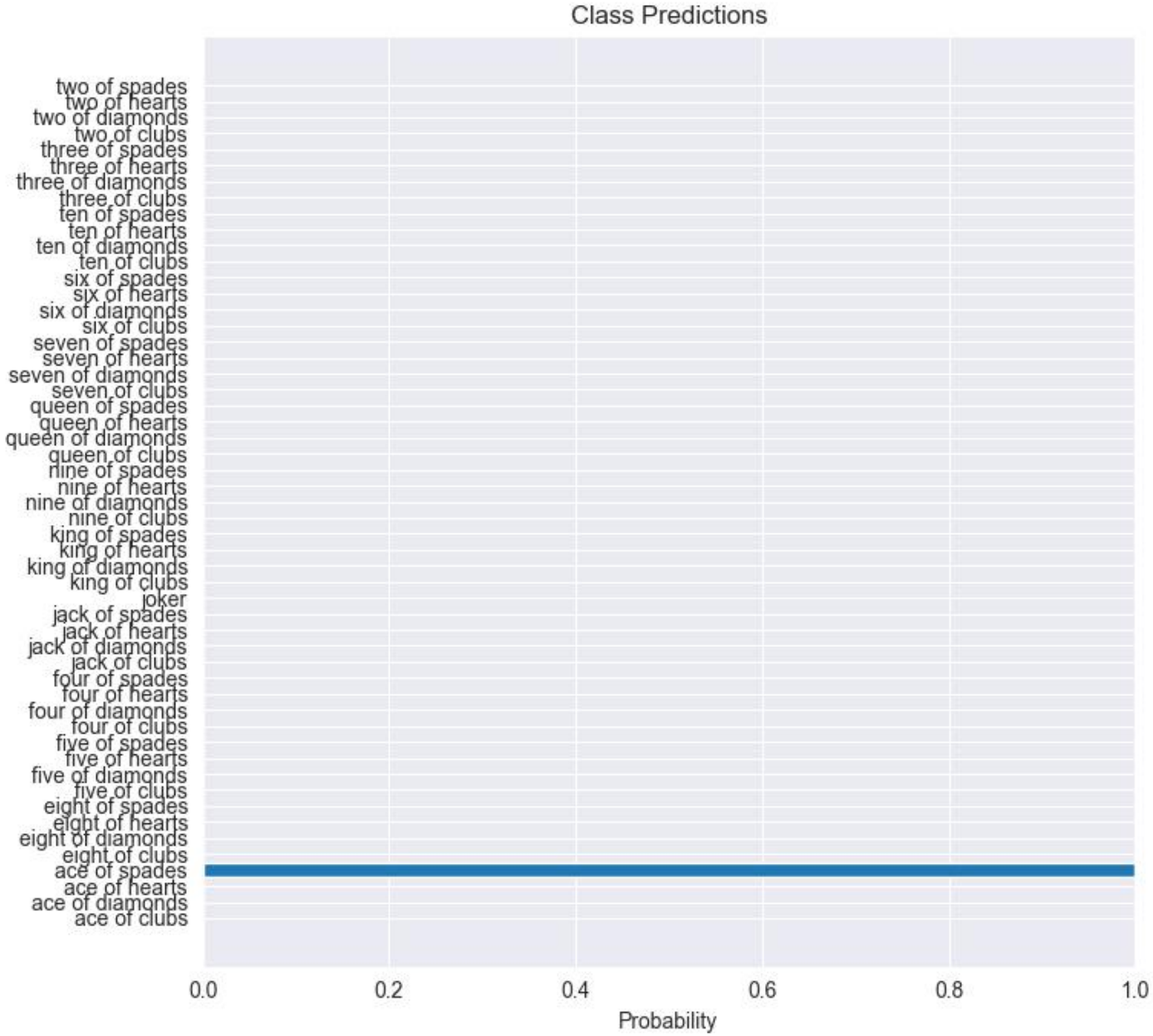
Ostatnim usprawnieniem było dodanie dodatkowej warstwy konwolucyjnej

```
class CardClassifier(nn.Module):
    def __init__(self, num_classes=53):
        super(CardClassifier, self).__init__()

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(128)

        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(256)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(256 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, num_classes)

        self.dropout1 = nn.Dropout(p=0.1)
        self.dropout2 = nn.Dropout(p=0.2)
        self.dropout3 = nn.Dropout(p=0.3)
    def forward(self, x):
        x = self.pool(F.gelu(self.bn1(self.conv1(x))))
        x = self.pool(F.gelu(self.bn2(self.conv2(x))))
        x = self.pool(F.gelu(self.bn3(self.conv3(x))))
        x = self.pool(F.gelu(self.bn4(self.conv4(x))))

        x = x.view(x.size(0), -1)
        x = F.gelu(self.fc1(x))
        x = self.dropout1(x)
        x = F.gelu(self.fc2(x))
        x = self.dropout2(x)
        x = F.gelu(self.fc3(x))
        x = self.dropout3(x)

        output = self.fc4(x)
        return output
```


Loss over epochs
Training loss
Validation loss
Loss
Epoch
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
10
20
30
40
50


Accuracy over epochs
Training accuracy
Validation accuracy
Accuracy
Epoch
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50


Confusion Matrix
True
Predicted
ace of clubs
ace of diamonds
ace of hearts
ace of spades
eight of clubs
eight of diamonds
eight of hearts
eight of spades
five of clubs
five of diamonds
five of hearts
five of spades
four of clubs
four of diamonds
four of hearts
four of spades
jack of clubs
jack of diamonds
jack of hearts
jack of spades
joker
king of clubs
king of diamonds
king of hearts
king of spades
nine of clubs
nine of diamonds
nine of hearts
nine of spades
queen of clubs
queen of diamonds
queen of hearts
queen of spades
seven of clubs
seven of diamonds
seven of hearts
seven of spades
six of clubs
six of diamonds
six of hearts
six of spades
ten of clubs
ten of diamonds
ten of hearts
ten of spades
three of clubs
three of diamonds
three of hearts
three of spades
two of clubs
two of diamonds
two of hearts
two of spades
0
1
2
3
4
5


Class Predictions
two of spades
two of hearts
two of diamonds
two of clubs
three of spades
three of hearts
three of diamonds
three of clubs
ten of spades
ten of hearts
ten of diamonds
ten of clubs
six of spades
six of hearts
six of diamonds
six of clubs
seven of spades
seven of hearts
seven of diamonds
seven of clubs
queen of spades
queen of hearts
queen of diamonds
queen of clubs
nine of spades
nine of hearts
nine of diamonds
nine of clubs
king of spades
king of hearts
king of diamonds
king of clubs
joker
jack of spades
jack of hearts
jack of diamonds
jack of clubs
four of spades
four of hearts
four of diamonds
four of clubs
five of spades
five of hearts
five of diamonds
five of clubs
eight of spades
eight of hearts
eight of diamonds
eight of clubs
ace of spades
ace of hearts
ace of diamonds
ace of clubs
0.0
0.2
0.4
0.6
0.8
1.0
Probability

WNIOSKI DO BADANIA 6

Sieć ta miała najlepsze wyniki.

- Dokładność 95.07%
- 43 epoki do przeuczenia

BADANIE 7

W siódmym badaniu postanowiono zmienić funkcje optymalizacyjne

Optimizer	Adam	AdamW	RMSprop	SGD
Accuracy	95.07%	93.7%	92.83%	85.66%

WNIOSKI DO BADANIA 7

Zmiana algorytmu optymalizacyjnego nie powoduje większych zmian w dokładności wyjątkiem jest SVD.

Adam charakteryzuje się najlepszą dokładnością.

BADANIE 8

W siódmy badaniu postanowiono zmienić proporcje zbioru uczącego zmniejszając zbiór testowy.

Proporcje	60-20-20	70-15-15	80-10-10	bazowy
Accuracy	78.19%	79.47%	85.73%	95.07%

WNIOSKI DO BADANIA 8

Zmniejszanie zbioru uczącego powoduje spadek dokładności sieciu

PODSUMOWANIE

- Drop-Out i Agumentacja w pozytywny sposób wpływają na czas uczenia i dokładność
- Dodatkowe warstwy wpływają w niewielki sposób na czas uczenia jednak zwiększają dokładność
- Bach-Normalisation i Weight-Decade w niewielkim stopniu wpływa na czas uczenia jednak zwiększają dokładność
- Zmniejszanie zbioru uczącego negatywnie wpływa na dokładność modelu.
- Algorytm optymalizacyjny Adam najlepiej wpływa na dokładność modelu