**Sztuczna Inteligencja 2023/24**

**Uczenie maszynowe – sieci neuronowe (laboratorium).**

**Błażej Domagała, 07.07.2024**

Do uczenia maszynowego metodą sieci neuronowych wybrano dataset **MNIST**, który zawiera dużą bazę obrazów cyfr zapisanych pismem ręcznym. Obrazy tam zawarte są podzielone na zbiór testowy i treningowy. Zbiór treningowy zawiera 60,000 obrazów, a zbiór testowy zawiera 10,000 obrazów. Dane są zapisane w są w postaci tensorów:

Piksele w tensorze reprezentowane są liczbami 0...255 (odcienie szarości). Poniżej wklejono 5 przykładowych obrazów z datasetu, otrzymanych za pomocą kodu:



Obraz zawierający biały, czarne, Grafika, symbol

Opis wygenerowany automatycznie

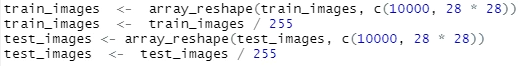
****

Do bazy dołączono wektor etykiet zawierający etykiety w postaci cyfr 0...9 oznaczających cyfrę zapisaną ręcznie w danym obrazie bazy.

Ze względu na ograniczone możliwości techniczne ograniczamy liczbę próbek w zbiorze treningowym do 10000 za pomocą kodu:



Do uczenia sieci neuronowej potrzebujemy wektorów, stąd zmniejszamy rank oryginalnego tensora z (10000, 28, 28) na (10000, 784), gdzie 10000 to liczba próbek, a 784 to liczba składowych wektora uczącego. Dodatkowo wskazane jest aby wartości poszczególnych składowych (pikseli) wektora uczącego zawierały się w przedziale [0, 1] co uzyskujemy dzieląc oryginalne wartości przez 255. Odpowiedni kod pokazano poniżej:



W celu zakwalifikowania obrazka do jednej z 10-ciu klas (0, 1, 2,...,9) w ostatniej warstwie umieszczono 10 jednostek („perceptronów”), gdzie każda jednostka będzie rozpoznawała określoną cyfrę. W związku z tym należy przekształcić oryginalny wektor etykiet (labels) do postaci tablicy [10000, 10], gdzie w każdym wierszu występuje dokładnie jedna jedynka - w kolumnie odpowiadającej jednostce rozpoznającej daną cyfrę. Wykorzystano do tego kod:



Poniżej przedstawiono 20 początkowych składowych oryginalnego wektora oraz „te same” składowe po przekształceniu „to\_categorical”:

> # Wyświetlenie oryginalnych etykiet dla pierwszych 20 próbek testowych

> original\_labels <- mnist$test$y[1:20]

> print(original\_labels)

[1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4

>

> # Wyświetlenie przekształconych etykiet dla tych samych próbek

> categorical\_labels <- test\_labels[1:20, 1:10]

> print(categorical\_labels)

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]

[1,] 0 0 0 0 0 0 0 1 0 0

[2,] 0 0 1 0 0 0 0 0 0 0

[3,] 0 1 0 0 0 0 0 0 0 0

[4,] 1 0 0 0 0 0 0 0 0 0

[5,] 0 0 0 0 1 0 0 0 0 0

[6,] 0 1 0 0 0 0 0 0 0 0

[7,] 0 0 0 0 1 0 0 0 0 0

[8,] 0 0 0 0 0 0 0 0 0 1

[9,] 0 0 0 0 0 1 0 0 0 0

[10,] 0 0 0 0 0 0 0 0 0 1

[11,] 1 0 0 0 0 0 0 0 0 0

[12,] 0 0 0 0 0 0 1 0 0 0

[13,] 0 0 0 0 0 0 0 0 0 1

[14,] 1 0 0 0 0 0 0 0 0 0

[15,] 0 1 0 0 0 0 0 0 0 0

[16,] 0 0 0 0 0 1 0 0 0 0

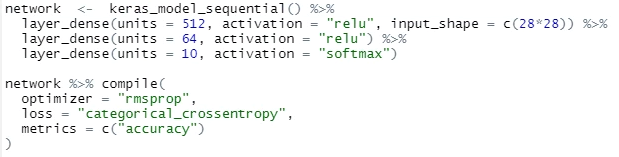
[17,] 0 0 0 0 0 0 0 0 0 1

[18,] 0 0 0 0 0 0 0 1 0 0

[19,] 0 0 0 1 0 0 0 0 0 0

[20,] 0 0 0 0 1 0 0 0 0 0

Dane zostały przygotowane do procesu tworzenia modelu sieci neuronowej. W następnym etapie tworzymy model sieci neuronowej oraz dokonujemy jego kompilacji. Przykładowy kod modelu z dwiema warstwami ukrytymi pokazano poniżej:



W naszych modelach wszystkie warstwy oprócz ostatniej wykorzystują funkcję aktywacji „relu”, natomiast ostatnia warstwa wykorzystuje funkcję „softmax”, gdyż chcemy otrzymać wyniki z przedziału [0,1]. Parametry kompilacji modelu dobrano kierując się tym, że celem naszym jest klasyfikacja na kategorie.

W celu przetestowania i porównania różnych sieci utworzono 5 modeli o parametrach podanych poniżej (2 modele z jedną warstwą ukrytą, 2 modele z dwiema warstwami, 1 model z trzema warstwami ukrytymi) :

Model 1

1. model1 <- keras\_model\_sequential() %>%

2. layer\_dense(units = 512, activation = 'relu', input\_shape = c(28 \* 28)) %>%

3. layer\_dense(units = 10, activation = 'softmax')

Model 2

1. model2 <- keras\_model\_sequential() %>%

2. layer\_dense(units = 256, activation = 'relu', input\_shape = c(28 \* 28)) %>%

3. layer\_dense(units = 10, activation = 'softmax')

Model 3

1. model3 <- keras\_model\_sequential() %>%

2. layer\_dense(units = 512, activation = 'relu', input\_shape = c(28 \* 28)) %>%

3. layer\_dense(units = 256, activation = 'relu') %>%

4. layer\_dense(units = 10, activation = 'softmax')

Model 4

1. model4 <- keras\_model\_sequential() %>%

2. layer\_dense(units = 256, activation = 'relu', input\_shape = c(28 \* 28)) %>%

3. layer\_dense(units = 128, activation = 'relu') %>%

4. layer\_dense(units = 10, activation = 'softmax')

Model 5

1. model5 <- keras\_model\_sequential() %>%

2. layer\_dense(units = 512, activation = 'relu', input\_shape = c(28 \* 28)) %>%

3. layer\_dense(units = 256, activation = 'relu') %>%

4. layer\_dense(units = 128, activation = 'relu') %>%

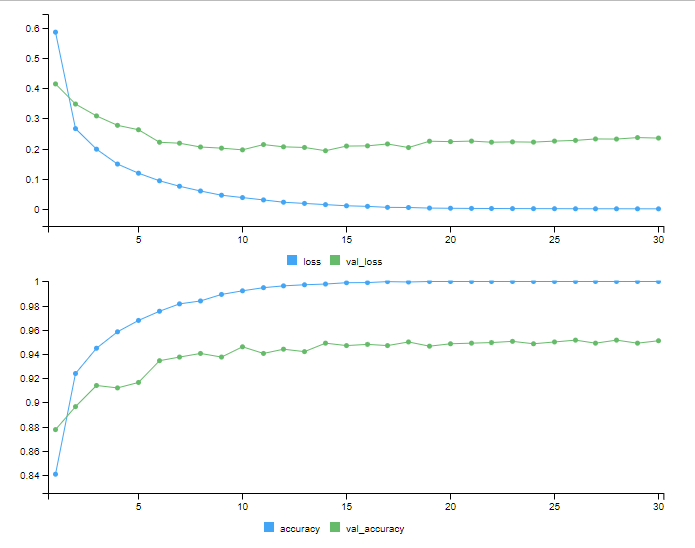
5. layer\_dense(units = 10, activation = 'softmax')

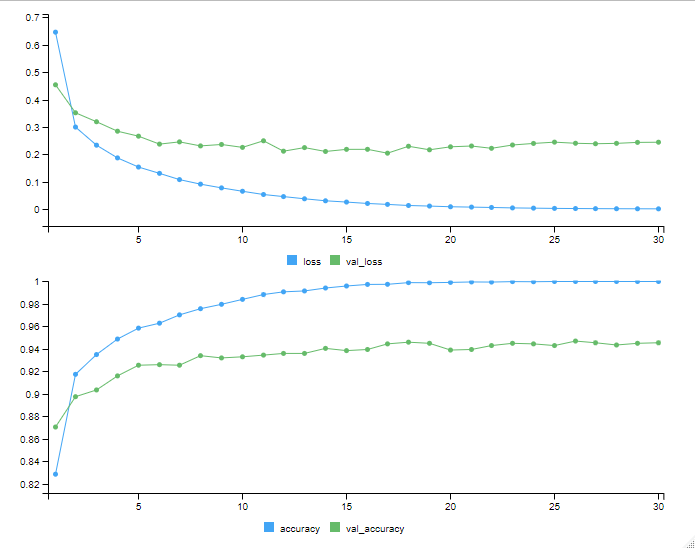
Następnie trenujemy modele za pomocą kodu:

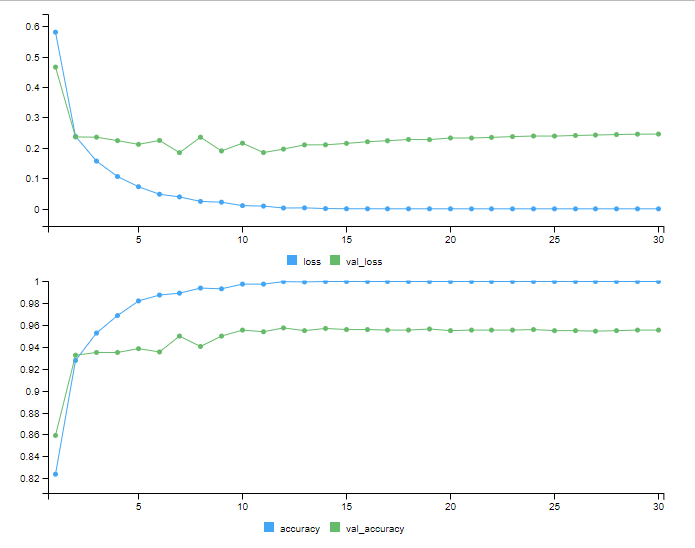


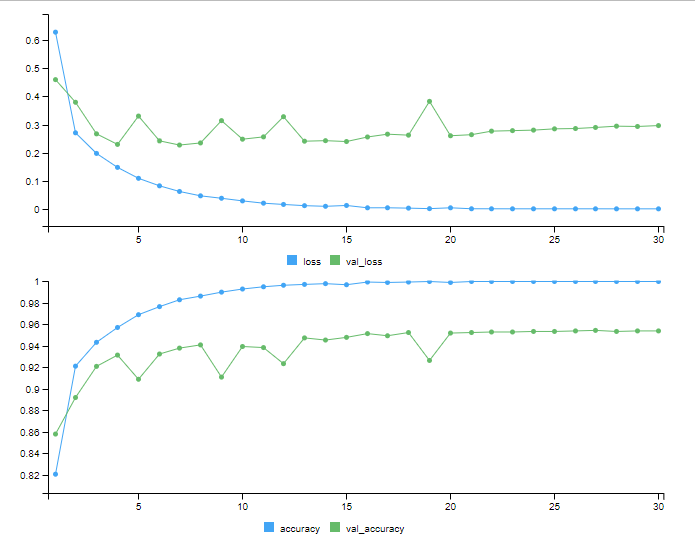
Rezerwujemy tutaj 20% zbioru treningowego do walidacji procesu uczenia. Hiperparametry epochs i batch\_size możemy modyfikować w razie potrzeby.

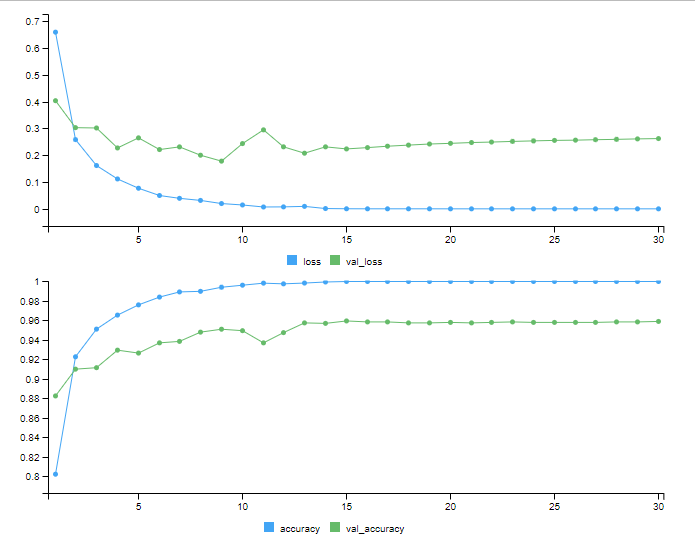
Poniżej przedstawiono wykresy błędu uczenia (loss) oraz dokładność rozpoznawania (accuracy) w trakcie uczenia dla części treningowej i walidacyjnej dla pięciu zaproponaowanych modeli.











Na podstawie wykresów zdecydowano się na ustalenie liczby epochs dla badanych modeli zgodnie z tabelką:

|  |  |
| --- | --- |
| Model nr | epochs |
| Model 1 | 20 |
| Model 2 | 25 |
| Model 3 | 30 |
| Model 4 | 25 |
| Model 5 | 30 |

Następnie dokonujemy ewaluacji zaproponowanych modeli na zbiorze testowym za pomocą kodu:



Wyniki przedstawiono w poniższej tabeli:

|  |  |  |
| --- | --- | --- |
| Model nr | loss | accuracy |
| Model 1 | 0.1854 | 0.9577 |
| Model 2 | 0.2053 | 0.9526 |
| Model 3 | 0.2052 | 0.9595 |
| Model 4 | 0.2265 | 0.9580 |
| Model 5 | 0.2227 | 0.9630 |

Na tej podstawie zdecydowano, że najlepszym z badanych modeli jest Model 5, dla którego accuracy wynosi 0.9630.

Zobaczmy, jak w przykładowych szczegółach prezentują się nasze wyniki. Dokonajmy predykcji i porównajmy wyniki z oryginalnymi etykietami reprezentującymi poprawne wartości dla 100 pierwszych liter:

> print(prediction\_result)

Predicted Actual

1 7 0

2 2 0

3 1 0

4 0 1

5 4 0

6 1 0

7 4 0

8 9 0

9 6 0

10 9 0

11 0 1

12 6 0

13 9 0

14 0 1

15 1 0

16 5 0

17 9 0

18 7 0

19 3 0

20 4 0

21 9 0

22 6 0

23 6 0

24 5 0

25 4 0

26 0 1

27 7 0

28 4 0

29 0 1

30 1 0

Następnie sprawdźmy, które litery z tych 100 zostały źle sklasyfikowane za pomocą kodu:

> print(wrong\_predictions)

Predicted Actual

1 7 0

2 2 0

3 1 0

4 0 1

5 4 0

6 1 0

7 4 0

8 9 0

9 6 0

10 9 0

11 0 1

12 6 0

13 9 0

14 0 1

15 1 0

16 5 0

17 9 0

18 7 0

19 3 0

20 4 0

21 9 0

22 6 0

23 6 0

24 5 0

25 4 0

26 0 1

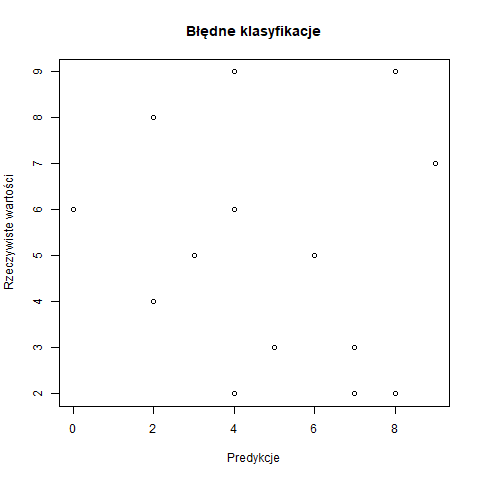
27 7 0

28 4 0

29 0 1

30 1 0

Wyniki przedstawiono poniżej:



Poniżej pokazano obrazki tych liter, które zostały źle sklasyfikowane:

1. # Błędnie sklasyfikowane próbki

2. wrong\_indices <- which(prediction\_result$Predicted != prediction\_result$Actual)

3. correct\_indices <- which(prediction\_result$Predicted == prediction\_result$Actual)

4.

5. # Wygeneruj obrazki dla błędnie sklasyfikowanych próbek

6. for (i in wrong\_indices) {

7. digit <- test\_images\_original[i,,]

8. png(filename = paste0("wrong\_digit\_", i, ".png"))

9. plot(as.raster(digit, max = 255))

10. dev.off()

11. }



Dla przykładu poniżej pokazano również trzy obrazki , które zostały prawidłowo rozpoznane

1. # Wygeneruj obrazki dla prawidłowo sklasyfikowanych próbek (3 przykłady)

2. for (i in correct\_indices[1:3]) {

3. digit <- test\_images\_original[i,,]

4. png(filename = paste0("correct\_digit\_", i, ".png"))

5. plot(as.raster(digit, max = 255))

6. dev.off()

7. }

