

KONWOLUCYJNE SIECI NEURONOWE

ORAZ PACZKA KERAS

KERAS DLA KLASYFIKACJI IRYSÓW

Paczka Scikit-Learn jest dobra do prostych zastosowań, ale w świecie poważniejszych badań korzysta się z paczek tensorflow i pytorch. Tensorflow ma w sobie dodatkowo dość fajny moduł/interfejs zwany keras. Wykorzystamy go dzisiaj do klasyfikacji.

Zacznijmy od prostszych rzeczy (plik załączony keras-iris.py):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Preprocess the data
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Encode the labels
encoder = OneHotEncoder(sparse=False)
y_encoded = encoder.fit_transform(y.reshape(-1, 1))

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.3,
random_state=42)

# Define the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(y_encoded.shape[1], activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

```

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")

# Plot the learning curve
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='validation accuracy')
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.grid(True, linestyle='--', color='grey')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.grid(True, linestyle='--', color='grey')
plt.legend()

plt.tight_layout()
plt.show()

# Save the model
model.save('iris_model.h5')

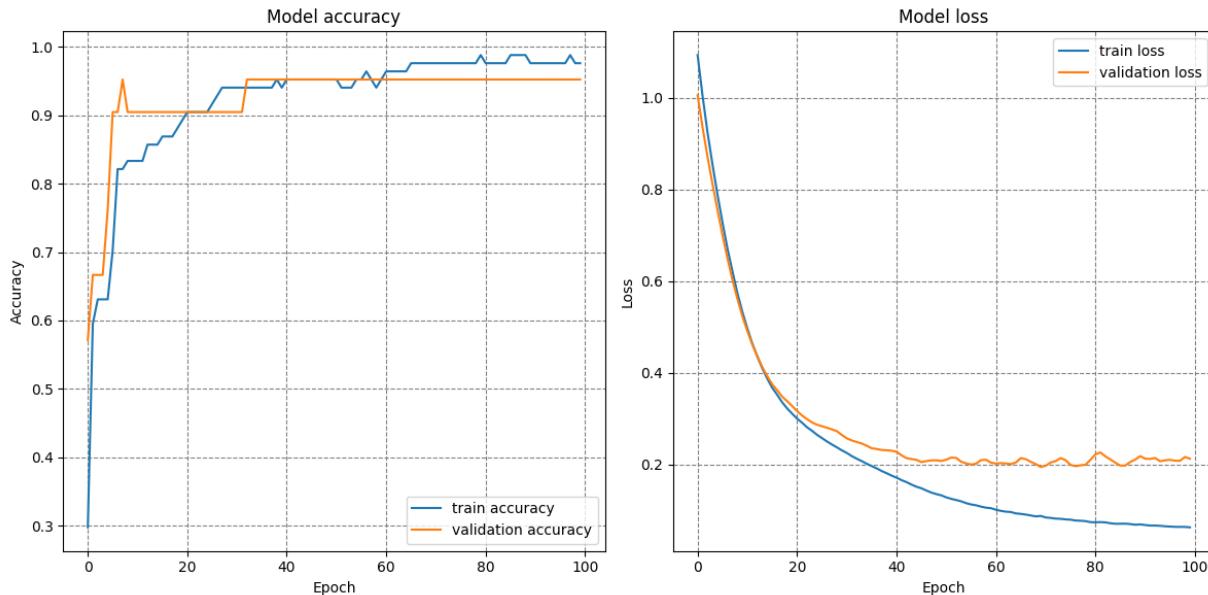
# Plot and save the model architecture
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

```

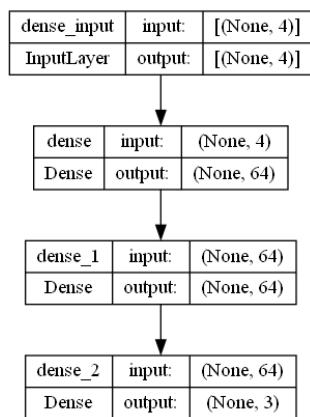
Program ten:

- Konstruuje sekwencyjny model sieci neuronowej.
- Uczy go na 100 epokach.
- Zbiera historię nauki i plotuje ją na krzywych uczenia się.
- Zapisuje model w pliku z rozszerzeniem h5.
- Zapisuje reprezentacje modelu w pliku model_plot.png

U mnie po nauce krzywe wyglądają tak:



Graficznie, model wygląda tak:



Przejdźmy do zadania.

ZADANIE 1: IRYSY W KERAS

Rozpatrz poniższe problemy i pytania. Odpowiedzi do nich możesz zapisać jako rozszerzone komentarze w pliku pythonowym. Wykonaj różne modyfikacje programu, żeby odpowiedzieć na pytania.

- Co robi StandardScaler? Jak transformowane są dane liczbowe?
- Czym jest OneHotEncoder (i kodowanie „one hot” ogólnie)? Jak etykiety klas są transformowane przez ten encoder?
- Model ma 4 warstwy: wejściową, dwie ukryte warstwy z 64 neuronami każda i warstwę wyjściową. Ile neuronów ma warstwa wejściowa i co oznacza `X_train.shape[1]`? Ile neuronów ma warstwa wyjściowa i co oznacza `y_encoded.shape[1]`?
- Czy funkcja aktywacji relu jest najlepsza do tego zadania? Spróbuj użyć innej funkcji i obejrzyj wyniki
- Model jest konfigurowany do treningu za pomocą polecenia `compile`. Tutaj wybieramy optymalizator (algorytm, który używa gradientu straty do aktualizacji wag), funkcję straty, metrykę do oceny modelu. Eksperymentuj ze zmianą tych parametrów na inne i uruchom program. Czy różne optymalizatory lub funkcje straty dają różne wyniki? Czy możemy dostosować szybkość uczenia się w optymalizatorze?
- W linii `model.fit` sieć neuronowa jest trenowana. Czy jest sposób, by zmodyfikować tę linię tak, aby rozmiar partii był równy 4 lub 8 lub 16? Jak wyglądają krzywe uczenia się dla różnych parametrów? Jak zmiana partii wpływa na kształt krzywych? Wypróbuj różne wartości i uruchom program.
- Co możesz powiedzieć o wydajności sieci neuronowej na podstawie krzywych uczenia? W której epoce sieć osiągnęła najlepszą wydajność? Czy ta krzywa sugeruje dobrze dopasowany model, czy mamy do czynienia z niedouczeniem lub przeuczeniem?
- Przejrzyj niżej wymieniony kod i wyjaśnij co się w nim dzieje.

```

import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from tensorflow.keras.models import load_model

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Preprocess the data
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Encode the labels
encoder = OneHotEncoder(sparse=False)
y_encoded = encoder.fit_transform(y.reshape(-1, 1))

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.3,
random_state=42)

# Load the pre-trained model
model = load_model('iris_model.h5')

# Continue training the model for 10 more epochs
model.fit(X_train, y_train, epochs=10)

# Save the updated model
model.save('updated_iris_model.h5')

# Evaluate the updated model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")

```

KERAS DLA KLASYFIKACJI OBRAZÓW

Użyjmy teraz keras do klasyfikacji cyfr (0, 1, 2, .., 9) na podstawie ręcznie pisanych reprezentacji obrazkowych. Załącznik: keras-mnist-cnn.py. Spójrzmy na kod:

```

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.keras.utils import to_categorical

```

```

from sklearn.metrics import confusion_matrix
from tensorflow.keras.callbacks import History

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess data
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
original_test_labels = np.argmax(test_labels, axis=1) # Save original labels for confusion matrix

# Define model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = History()
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_split=0.2,
callbacks=[history])

# Evaluate on test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc:.4f}")

# Predict on test images
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

# Confusion matrix
cm = confusion_matrix(original_test_labels, predicted_labels)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Plotting training and validation accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)

```

```

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True, linestyle='--', color='grey')
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True, linestyle='--', color='grey')
plt.legend()

plt.tight_layout()
plt.show()

# Display 25 images from the test set with their predicted labels
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i].reshape(28,28), cmap=plt.cm.binary)
    plt.xlabel(predicted_labels[i])
plt.show()

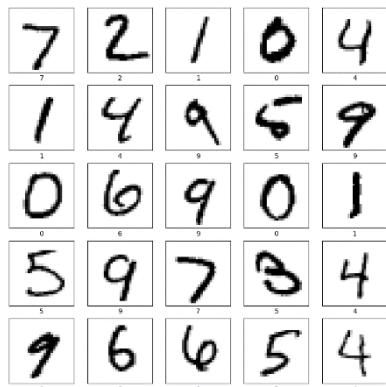
```

Dla powyższego kodu rozwiążmy poniższe zadanie:

ZADANIE 2: KERAS I CYFRY NA OBRAZACH

Odpowiedz na pytania (jako komentarze w programie). Do niektórych odpowiedzi musisz uruchomić program z różnymi modyfikacjami.

- Co się dzieje w preprocessing? Do czego służy funkcja reshape, to_categorical i np.argmax?
- Jak dane przepływają przez sieć i jak się w niej transformują? Co każda z warstw dostaje na wejście i co wyrzuca na wyjściu?
- Jakich błędów na macierzy błędów jest najwięcej. Które cyfry są często mylone z jakimi innymi?
- Co możesz powiedzieć o krzywych uczenia się. Czy mamy przypadek przeuczenia lub niedouczenia się?
- Jak zmodyfikować kod programu, aby model sieci był zapisywany do pliku h5 co epokę, pod warunkiem, że w tej epoce osiągnęliśmy lepszy wynik?



Ostatnie zadanie dotyczy już zewnętrznej bazy danych.

ZADANIE 3: PSY I KOTY

Celem zadania jest wykorzystanie sieci konwolucyjnych do rozpoznawania psów i kotów z obrazków. Paczka ze zdjęciami (miniaturki) jest w załączniku (*dogs-cats-mini.zip*) i zwiera dokładnie 12500 zdjęć kotów i 12500 zdjęć psów (razem około 26 MB).

Jaki jest plan działania:

- a) Zapoznaj się z wybranym samouczkiem np.
<https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/> lub
<https://www.kaggle.com/code/seysimty/keras-cnn-dog-or-cat-classification> lub inne.
- b) Załaduj bazę danych i dokonaj jej obróbki (przetworzenie obrazów, wyciągnięcie klasy cat/dog z nazwy obrazka, itd.).
- c) Skonstruuj, wytrenuj model sieci konwolucyjnej na zbiorze treningowym. Sieć może mieć standardową, zaproponowaną w Internecie konfigurację.
- d) Dokonaj walidacji. Podaj krzywą uczenia się dla zbioru treningowego i walidacyjnego.
(Pytania dodatkowe: Czy są koty przypominające psy, albo psy przypominające koty? Ile? Potrafisz może wskazać konkretne zdjęcia omyłkowo zakwalifikowanych zwierząt?)
- e) Powtórz parę razy eksperyment c-d z inną konfiguracją sieci (optimizer, funkcje aktywacji, inna struktura sieci, dropout, itp.). Wskaż jaka konfiguracja działała najlepiej i pokaż jej wyniki (krzywa uczenia się, dokładność, macierz błędu).
(W razie długiego treningu pamiętaj o możliwości zapisywania sieci w pliku i dotrenowania później).

