Coman Ruxandra (@blankusus)

Convolutional Neural Network for Hiragana Recognition

Introduction

This script trains a Convolutional Neural Network (CNN) on a dataset of 28x28 images of hiragana characters for multi-class recognition. The dataset is imported from Kaggle (see references) and it consists of 10 classes, a training set which contains 60,000 images and a testing set, containing 10,000 images. After training, the accuracy score is printed and also, an image of the first 25 characters in the dataset, the confusion matrix and plots of training and validation loss and accuracy over the training epochs are present.

Libraries

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

- **Pandas** – used for loading the training and test datasets from CSV files, dropping the unnecessary columns and extracting features and labels from the given data;
- **Numpy** – used for reshaping the data into the wanted format and also normalizing the pixel values;
- **Tensorflow and keras** – used for defining the CNN, compiling, training and evaluating the model and using callbacks for early stopping and saving the best model;
- **Matplotlib** – plotting sample images from the training set and training and validation loss and accuracy over epochs;
- **Sklearn** – computing the confusion matrix;
- **Seaborn** – visualizing the confusion matrix with a heatmap.

22.05.2024

Accessing the datasets

```
train_data =
pd.read_csv(r"C:\Users\Ruxi\PycharmProjects\pythonProject14\train.csv")
test_data =
pd.read_csv(r"C:\Users\Ruxi\PycharmProjects\pythonProject14\test.csv"
```

After placing the train and test datasets into a specific directory and introducing the path in the code, both the training and test data are loaded.

Unnamed columns

```
train_data = train_data.drop('Unnamed: 0', axis=1)
test_data = test_data.drop('Unnamed: 0', axis=1)
```

Because in both datasets the first column contains the labels, I had to drop it in order not to interfere with the rest of the data.

Images modifications

```
X_train = train_data.iloc[:, 1:].values.reshape(-1, 28, 28, 1)
y_train = train_data.iloc[:, 0].values

X_test = test_data.iloc[:, 1:].values.reshape(-1, 28, 28, 1)
y_test = test_data.iloc[:, 0].values

X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

I reshaped the training and test data to 28x28 images with 1 channel and grayscale, then extracted the labels. Additionally, I normalized the pixels by dividing each of them by 255.0 to scale them from [0, 255] to [0.0, 1.0].

Model architecture

```
model = models.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, (3, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
model_checkpoint = callbacks.ModelCheckpoint('best_model.keras',
save_best_only=True, monitor='val_loss')

history = model.fit(X_train, y_train, epochs=10, validation_split=0.2,
callbacks=[early_stopping, model_checkpoint])
```

- **Conv2D** – these layers apply some convolutional filters to the images to detect features such as edges, textures and patterns (32, 64, 128 filters);
- **MaxPooling2D** – these layers help with reducing the spatial dimensions of the feature maps resulting in reduced computational load;
- **Flatten** – this layer flattens the 3D output of the last convolutional layer to a 1D vector;
- **Dense** – these layers are fully connected and their goal is to classify images based on the features extracted from the convolutional layers (256, 128, 10 units).

The **model.compile** function configures the model for training.

- **EarlyStopping** - stops training if validation loss does not improve for 3 epochs;
- **ModelCheckpoint** - saves the best model based on validation loss.

The callbacks help prevent overfitting while saving the best version of the model during training.

The **model.fit** function trains the model on the training data for a specified number of epochs and validates it using only a portion of the training data, in my case 20% of the training data will be used.

Accuracy score

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

This evaluates the model on the test data to estimate its performance and then prints the test accuracy.

These are my results after 10 epochs:

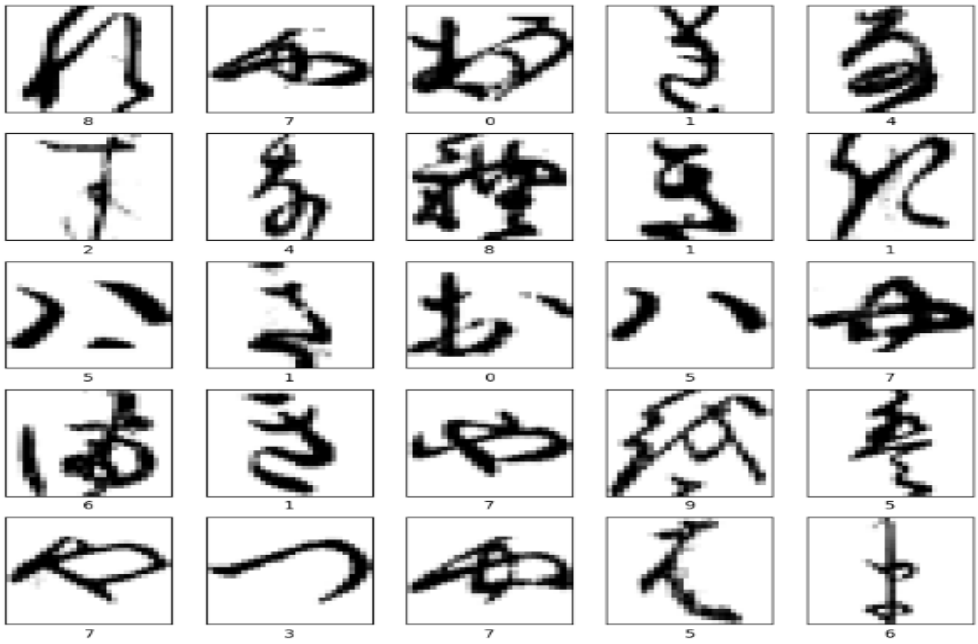
```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23s 14ms/step - accuracy: 0.7489 - loss: 0.7525 - val_accuracy: 0.9326 - val_loss: 0.2185
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 14ms/step - accuracy: 0.9502 - loss: 0.1621 - val_accuracy: 0.9501 - val_loss: 0.1653
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18s 12ms/step - accuracy: 0.9687 - loss: 0.1014 - val_accuracy: 0.9624 - val_loss: 0.1243
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18s 12ms/step - accuracy: 0.9769 - loss: 0.0709 - val_accuracy: 0.9596 - val_loss: 0.1389
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.9822 - loss: 0.0561 - val_accuracy: 0.9695 - val_loss: 0.1186
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.9866 - loss: 0.0422 - val_accuracy: 0.9721 - val_loss: 0.1096
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20s 14ms/step - accuracy: 0.9881 - loss: 0.0381 - val_accuracy: 0.9635 - val_loss: 0.1470
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18s 12ms/step - accuracy: 0.9891 - loss: 0.0338 - val_accuracy: 0.9670 - val_loss: 0.1355
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24s 14ms/step - accuracy: 0.9917 - loss: 0.0257 - val_accuracy: 0.9673 - val_loss: 0.1350
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9249 - loss: 0.3116
Test accuracy: 0.9277999997138977
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
```

First plot

```
img = [str(i) for i in range(10)]

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(X_train[i].reshape((28, 28)), cmap=plt.cm.binary)
    plt.xlabel(img[y_train[i]])
```

This creates a 5x5 grid of subplots and displays the first 25 images from the training dataset. Each image is shown in grayscale and has the corresponding digit label.



First 25 images from the training dataset

Second plot

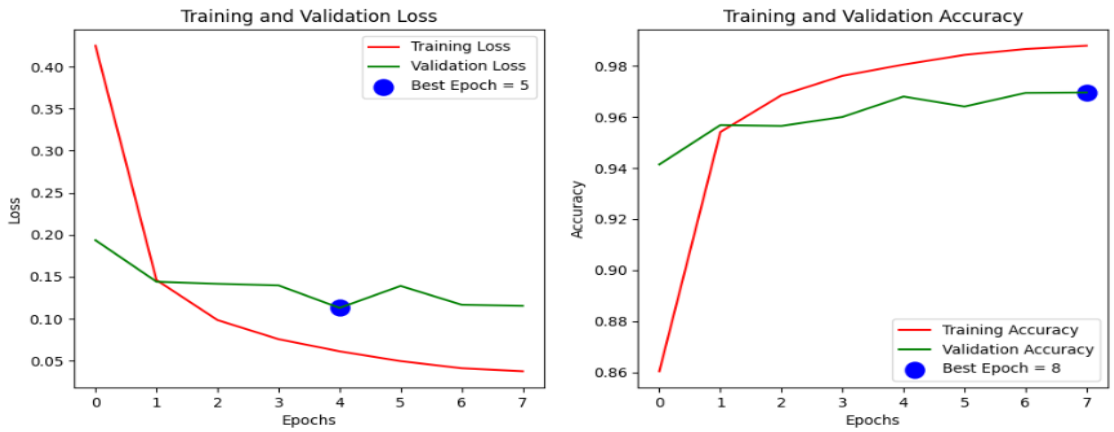
```
tacc = history.history['accuracy']
tloss = history.history['loss']
vacc = history.history['val_accuracy']
vloss = history.history['val_loss']

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(tloss, label='Training Loss', color='red')
plt.plot(vloss, label='Validation Loss', color='green')
plt.scatter(np.argmin(vloss), np.min(vloss), s=150, c='blue', label=f'Best
Epoch = {np.argmin(vloss) + 1}')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(tacc, label='Training Accuracy', color='red')
plt.plot(vacc, label='Validation Accuracy', color='green')
plt.scatter(np.argmax(vacc), np.max(vacc), s=150, c='blue', label=f'Best
Epoch = {np.argmax(vacc) + 1}')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

This plot is for visualizing the training and validation loss and accuracy over the epochs. The training loss and accuracy provide details about how well the model is fitting the training data. The validation loss and accuracy provide details about how well the model is performing on data that the model has not been exposed to during the training. By marking the best epoch, we know the point at which the model performed the best during training.



Training and validation loss and accuracy plot

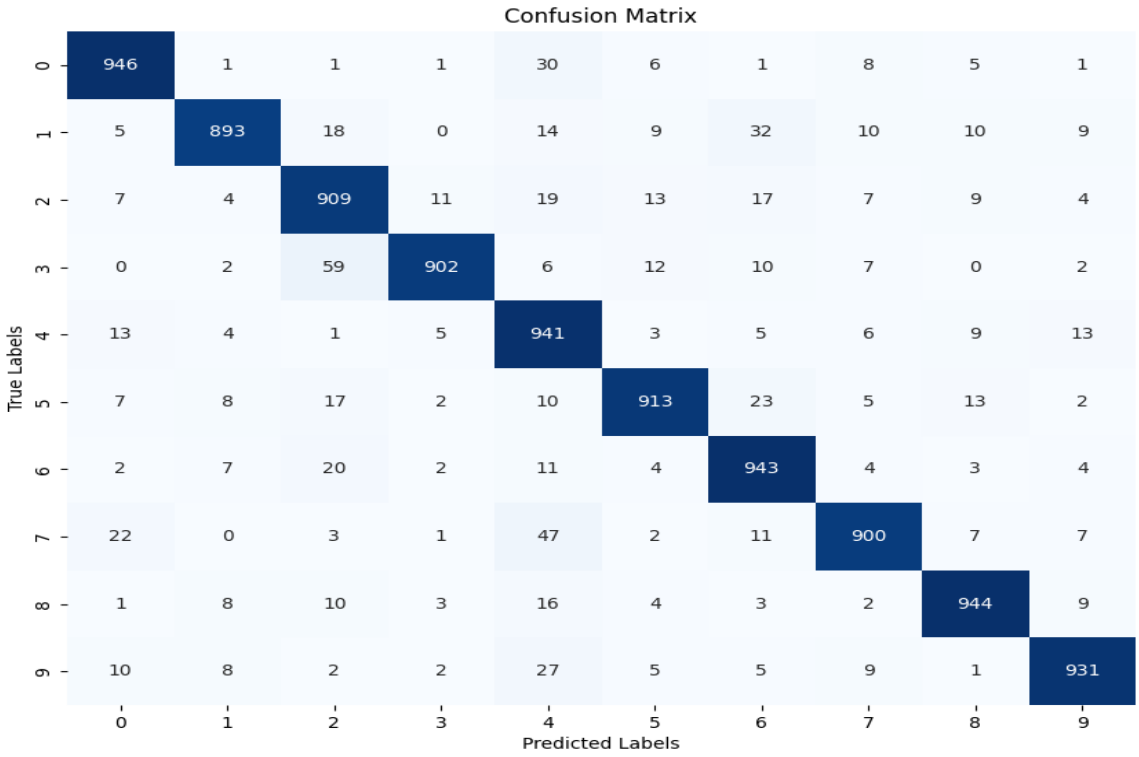
22.05.2024

Third plot

```
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

This computes the confusion matrix based on the model's predictions on the test data and sees it as a heatmap. It helps with identifying misclassifications.



Confusion matrix

22.05.2024

Conclusion

The provided code can be a "guide" to training a CNN for multi-class recognition tasks on hiragana character images. By using callbacks, the model's performance during training can be monitored and also stop the training if that performance starts to degrade. The dataset's origin, preprocessing steps, model architecture, training process and results are all outlined by it. Overall, I am pleased with this project and I believe that it does its job accordingly.

References

1. https://www.kaggle.com/datasets/notshrirang/japanese-characters/data
   (Hiragana Character Dataset, Shrirang Mahajan, 2023*)*
2. https://www.kaggle.com/code/notshrirang/sample-notebook-for-hiragana-classification
   (*Sample notebook for hiragana classification*, Shrirang Mahajan, 2023)
3. https://www.kaggle.com/code/gpiosenka/cnn-small-model-f1-score-97 (*CNN small model*, Gerry, 2023)
4. https://curs.upb.ro/2023/pluginfile.php/281800/mod_resource/content/1/Neural%20Networks%20and%20Genetic%20Algorithms%20Laboratory%20Exercises.pdf
   (*NNGA laboratory guide*, Prof.dr.ing. Ionel-Bujorel Pavaloiu, 2022)

22.05.2024