



Projekt zaliczeniowy - Rozpoznawanie hieroglifów

Autorzy: J. Klamra, M. Kida, K. Król

Politechnika Krakowska
Przetwarzanie Języka Naturalnego
2 czerwca 2024

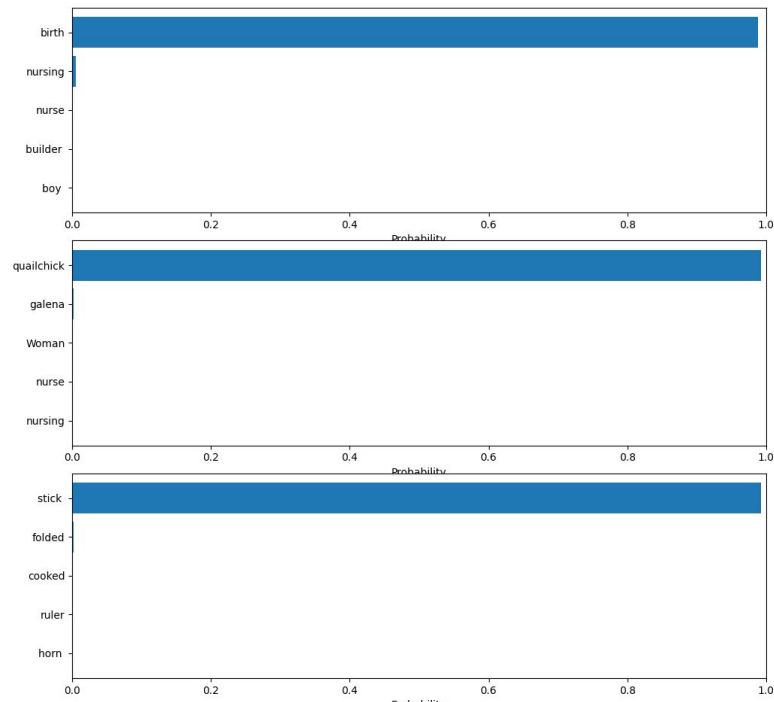
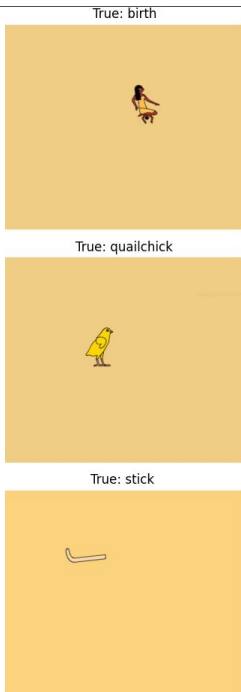


Plan

1. Abstrakt
2. Wstęp
 - 2.1. Cel projektu
 - 2.2. Zakres pracy
 - 2.3. Użyta metodyka
3. Wiedza potrzebna do rozpoznawania hieroglifów
4. Przedstawienie rozwiązania - kodu
5. Podsumowanie
6. Bibliografia

Cel projektu

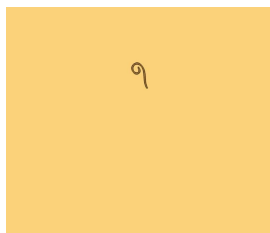
Celem niniejszego projektu było stworzenie programu potrafiącego poprawnie odczytywać hieroglify ideograficzne oraz zdekodowanie ich na język angielski.



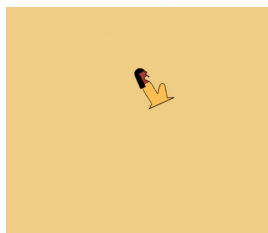
Zakres pracy

Do projektu został użyty zbiór danych *Egyptian Hieroglyphics Dataset* ze strony <https://www.kaggle.com/datasets/waleedumer/egyptian-hieroglyphics-datasets>. Posiada on 7779 plików do wytrenowania modelu oraz 2100 plików testowych, na których sprawdzane jest, czy model poprawnie rozpoznaje kształty.

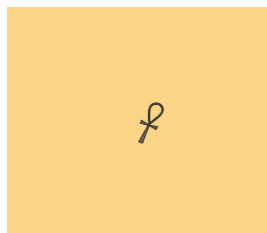
Poniżej przedstawione zostały przykładowe hieroglify.



100



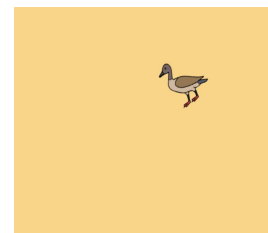
woman



ankh



enclosed
mound



duck



Użyta metodyka

W projekcie wykorzystane zostały następujące procesy:

1. Przygotowanie zbioru danych obejmujące poniższe elementy:
 - a. Zgrupowanie danych w katalogi szkoleniowe i testowe.
 - b. Implementacja klasy *ImageDataset* do obsługi ładowania obrazów, kodowania etykiet i transformacji.
 - c. Wyodrębnienie i kodowanie etykiet za pomocą *LabelEncoder*.
 - d. Przekształcanie obrazów przy użyciu transformacji powiązanych ze wstępnie wytrenowanym modelem *ResNet-50*.
2. Trening modelu - użycie wstępnie wytrenowanego modelu *ResNet-50* jako modelu bazowego. Na proces uczenia składa się:
 - a. Wykorzystanie pętli szkoleniowej przy określonej ilości epok, korzystając z optymalizatora *Adam* i funkcji utraty entropii krzyżowej. Przy każdej iteracji parametry były aktualizowane w oparciu o gradienty obliczone na podstawie strat.
 - b. Na koniec każdej iteracji obliczane i drukowane były straty w celu śledzenia postępów modelu i dopasowywania parametrów.



Użyta metodyka - c.d.

3. Ocena - po zakończeniu uczenia model zostawał poddawany ocenie:
 - a. Wydajność modelu została oceniona na podstawie testowego zestawu danych.
 - b. Dokładność została obliczona przez porównanie przewidywanych etykiet z prawdziwymi.
 - c. Błędnie sklasyfikowane obrazy zostały zidentyfikowane i zapisane do dalszej analizy.
4. Wizualizacja - w celu łatwiejszego wglądu w uzyskane efekty uczenia, zaimplementowane zostały następujące metody wizualizacji:
 - a. Pokazanie zestawu błędnie sklasyfikowanych obrazów z ich prawdziwymi i przewidywanymi etykietami.
 - b. Wyświetlanie losowych obrazów z zestawu testowego wraz z ich prawdziwymi etykietami i najlepszymi przewidywanymi przez model, wraz z prawdopodobieństwem.



Wiedza potrzebna do rozpoznawania hieroglifów

Znaki w piśmie hieroglificznym dzielą się na fonetyczne, ideograficzne i determinatywne. Ich prawidłowe rozpoznanie jest kluczowe w zrozumieniu przekazywanej treści.

Projekt skupia się na znakach ideograficznych, które wskazują na konkretne pojęcia, a nie litery.

Uczenie maszynowe może być przydatnym narzędziem do zrozumienia hieroglifów - za pomocą algorytmów zdolnych do samodzielnego uczenia się na podstawie dostarczonych danych możliwe jest uzyskanie wysoce prawdopodobnych wyników. Konieczne jest jednak użycie odpowiednich technik statystycznych i optymalizacyjnych, aby model mógł rozpoznać wzorce i sklasyfikować obrazy.

W projekcie wykorzystane zostały biblioteki:

- *PyTorch* - dostarczająca zestaw narzędzi do pracy z tensorami, a także definiowania modeli sieci neuronowych i operacji optymalizacyjnych.
- *Torchvision* - pakiet biblioteki *PyTorch*, zawierający narzędzia do pracy z zestawami danych obrazowych oraz zestawy narzędzi do przetwarzania obrazów.
- *Scikit-learn* i *scikit-image* - do konwersji etykiet do postaci liczbowej oraz wczytywania i zapisywania obrazów.



Przedstawienie rozwiązania i omówienie kodu

Klasa *ImageDataset* została zaimplementowana do przetwarzania własnych zestawów danych złożonych z obrazów. Jest rozszerzeniem klasy *Dataset* z biblioteki *Py-torch*. W konstruktorze odczytujemy pliki JPG z katalogu. Następnie zapisywane są ich ścieżki i kodowane etykiety za pomocą *LabelEncoder* z biblioteki *scikit-learn*.

Metoda *__getitem__()* zwraca obraz z możliwą jego transformacją.

```
19 class ImageDataset(Dataset):
20     def __init__(self, directory, transform=None):
21         self.directory = directory
22         self.transform = transform
23         self.image_paths = []
24         self.labels = []
25
26         for file_name in os.listdir(directory):
27             if file_name.endswith('.jpg'):
28                 label = file_name.split('.')[0]
29                 self.image_paths.append(os.path.join(directory, file_name))
30                 self.labels.append(label)
31
32         self.label_encoder = LabelEncoder()
33         self.labels = self.label_encoder.fit_transform(self.labels)
34
35     def __len__(self):
36         return len(self.image_paths)
37
38     def __getitem__(self, idx):
39         image_path = self.image_paths[idx]
40         image = read_image(image_path)
41         label = self.labels[idx]
42
43         if self.transform:
44             image = self.transform(image)
45
46         return image, label, image_path
47
```




Przedstawienie rozwiązania i omówienie kodu

W kolejnej części następuje inicjalizacja wstępnie wytrenowanego modelu *ResNet-50*, używając domyślnych dla niego wag. Model ustawiany jest w tryb ewaluacji.

Inicjalizowane są obiekty klas *ImageDataset* oraz *DataLoader*. Obiekty są osobne dla zbiorów danych treningowych i testowych.

Ostatnia, wynikowa warstwa modelu składa się z liczby neuronów równej liczbie osobnych klas hieroglifów z naszego zbioru danych.

W celu obliczenia strat wykorzystana zostanie funkcja *CrossEntropyLoss()*, a do optymalizacji - optymalizator *Adam* z współczynnikiem uczenia równym 0.00005.

```
47
48 weights=ResNet50_Weights.DEFAULT
49
50
51 model = models.resnet50(weights=weights)
52 model.eval()
53
54 transform = weights.transforms()
55
56 train_dataset = ImageDataset(directory='Train', transform=transform)
57 test_dataset = ImageDataset(directory='Test', transform=transform)
58
59 train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
60 test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
61
62 num_features = model.fc.in_features
63 model.fc = nn.Linear(num_features, len(train_dataset.label_encoder.classes_))
64 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
65 model = model.to(device)
66
67 criterion = nn.CrossEntropyLoss()
68 optimizer = optim.Adam(model.parameters(), lr=0.00005)
69
```

Przedstawienie rozwiązania i omówienie kodu

Funkcja *train()* używana jest do trenowania modelu. Dla danej liczby epok (11) wykonywane są następujące kroki:

- ustawienie modelu w tryb uczenia,
- iteracja przez bloki danych:
 - reset gradientu optymalizatora,
 - przejście przez model w celu uzyskania predykcji,
 - obliczanie straty i przejście wsteczne, obliczenie gradientu,
 - aktualizacja parametrów modelu na podstawie gradientu,
 - akumulacja łącznej straty,
- ustawienie modelu w tryb ewaluacji,
- przejście przez bloki danych i odczyt % poprawnych predykcji
- zapisanie modelu do pliku

```
70 def train():
71
72     num_epochs = 11
73
74     for epoch in range(num_epochs):
75         model.train()
76         running_loss = 0.0
77
78         for images, labels in train_loader:
79             images, labels = images.to(device), labels.to(device)
80             #transform here instead
81             optimizer.zero_grad()
82
83             outputs = model(images)
84             loss = criterion(outputs, labels)
85             loss.backward()
86             optimizer.step()
87
88             running_loss += loss.item()
89
90         print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}')
91
92     model.eval()
93     correct = 0
94     total = 0
95
96     with torch.no_grad():
97         for images, labels, _ in test_loader:
98             images, labels = images.to(device), labels.to(device)
99             outputs = model(images)
100             _, predicted = torch.max(outputs.data, 1)
101             total += labels.size(0)
102             correct += (predicted == labels).sum().item()
103
104     print(f'Test Accuracy: {100 * correct / total:.2f}%)
105
106     torch.save(model.state_dict(), 'model.pth')
107
```

Przedstawienie rozwiązania i omówienie kodu

Funkcja `test_and_show_misclassifications()` odczytuje zapisany model i ustawia go w tryb ewaluacji.

Następnie wykonywane jest przejście przez zestaw danych i sprawdzenie, czy przewidziana etykieta odpowiada właściwej. W przypadku nieodpowiedniego wyniku przypadek zapisywany jest jako niedopasowanie.

W kolejnym kroku niedopasowania są wyświetlane graficznie, za pomocą biblioteki *matplotlib*, po uprzedniej odwrotnej transformacji w celu uzyskania etykiet.

```
108 def test_and_show_misclassifications(model, test_loader, label_encoder, device):
109     model.load_state_dict(torch.load('model.pth'))
110     model.eval()
111     misclassified_images = []
112     misclassified_labels = []
113     misclassified_preds = []
114
115     with torch.no_grad():
116         for images, labels, _ in test_loader:
117             images, labels = images.to(device), labels.to(device)
118             outputs = model(images)
119             _, predicted = torch.max(outputs.data, 1)
120
121             for i in range(images.size(0)):
122                 if predicted[i] != labels[i]:
123                     misclassified_images.append(images[i].cpu())
124                     misclassified_labels.append(labels[i].cpu())
125                     misclassified_preds.append(predicted[i].cpu())
126
127     num_misclassifications = len(misclassified_images)
128     print(f'Number of misclassifications: {num_misclassifications}')
129
130     # Display some misclassified images
131     if num_misclassifications > 0:
132         fig, axs = plt.subplots(2, 5, figsize=(15, 6))
133         for i in range(10):
134             if i >= num_misclassifications:
135                 break
136             ax = axs[1 // 5, i % 5]
137             img = F.to_pil_image(misclassified_images[i])
138             true_label = label_encoder.inverse_transform([misclassified_labels[i].item()])[0]
139             pred_label = label_encoder.inverse_transform([misclassified_preds[i].item()])[0]
140             ax.imshow(img)
141             ax.set_title(f'True: {true_label}\nPred: {pred_label}')
142             ax.axis('off')
143         plt.tight_layout()
144         plt.show()
145
```

Przedstawienie rozwiązania i omówienie kodu

Funkcja `show_random_predictions()` wykorzystana została do wybrania kilku obrazów z testowego zestawu danych, dokonania predykcji wykorzystując wytrenowany model, a następnie do wyświetlenia na wykresie najbardziej prawdopodobnych wyników.

Na początku model jest odczytywany i ustawiony w tryb ewaluacji, a zdefiniowana ilość próbek jest wybierana z zestawu. Dla każdej próbki odczytywany jest tensor obrazu i jego pozostałe parametry. Wykonywane jest przejście przez model w celu uzyskania predykcji, następnie obliczane są prawdopodobieństwa przy pomocy funkcji `softmax()`. Wektor prawdopodobieństwa jest przeniesiony do procesora i przekonwertowany na tablicę, wybierając pierwszy element.

W kolejnym kroku uzyskane wyniki są sortowane malejąco, transformacja odwrotna pozwala na uzyskanie etykiet, a 5 najlepszych predykcji jest wyświetlana graficznie.

```
147 def show_random_predictions(model, test_dataset, label_encoder, device, num_images=3):
148     model.load_state_dict(torch.load('model.pth'))
149     model.eval()
150     indices = random.sample(range(len(test_dataset)), num_images)
151
152     fig = plt.figure(figsize=(15, 5 * num_images))
153     gs = gridspec.GridSpec(num_images, 2, width_ratios=[1, 1.5])
154
155     with torch.no_grad():
156         for i, idx in enumerate(indices):
157             _, label, image_path = test_dataset[idx]
158             image_tensor = _.unsqueeze(0).to(device) # Add batch dimension and move to device
159
160             outputs = model(image_tensor)
161             probabilities = nn.functional.softmax(outputs, dim=1)
162             probabilities = probabilities.cpu().numpy()[0] # Move to CPU and get the first item
163
164             sorted_indices = np.argsort(-probabilities) # Sort in descending order
165             sorted_probabilities = probabilities[sorted_indices]
166             sorted_labels = label_encoder.inverse_transform(sorted_indices)
167
168             # Show original image
169             ax_img = plt.subplot(gs[i, 0])
170             original_image = io.imread(image_path)
171             ax_img.imshow(original_image)
172             ax_img.axis('off')
173
174             true_label = label_encoder.inverse_transform([label])[0]
175             ax_img.set_title(f"True: {true_label}")
176
177             # Show top predictions as bar graph
178             ax_bar = plt.subplot(gs[i, 1])
179             top_predictions = {sorted_labels[j]: sorted_probabilities[j] for j in range(5)}
180
181             labels = list(top_predictions.keys())
182             probs = list(top_predictions.values())
183             ax_bar.barh(labels, probs)
184             ax_bar.set_xlim(0, 1)
185             ax_bar.set_xlabel('Probability')
186             ax_bar.invert_yaxis() # To have the highest probability on top
187
188     plt.tight_layout()
189     plt.show()
```



Podsumowanie

- Cel projektu został osiągnięty.
- System poprawnie tłumaczy losowe obrazy.
- Program można w łatwy sposób rozbudować o kolejne znaki, które będą rozpoznawane.
- Do rozpoznawania hieroglifów nie trzeba koniecznie używać biblioteki scikit-learn.
- Model może z powodzeniem zostać użyty do rozpoznawania różnych obrazów. Wystarczy przygotować odpowiedni zbiór danych uczących.



Bibliografia

<https://www.kaggle.com/datasets/waleedumer/egyptian-hieroglyphics-datasets> - dostęp 30.05.2024

<https://ii.pk.edu.pl/~rkycia/classes/2015/files/Projekty/Hieroglify/HieroglifyPrezentacja.pdf> - dostęp 30.05.2024

<https://pytorch.org/> - dostęp 30.05.2024

https://pl.wikipedia.org/wiki/Pismo_hieroglificzne - dostęp 30.05.2024