Name, programme

- Nahusenay Yifter MPALG
- Erik Nilsson MPALG

E-mail address

- yifter@student.chalmers.se
- ernilsso@student.chalmers.se

The number of hours spent working on the assignment for each one of you (excluding lectures)

- Nahusenay: 20+ hours
- Erik: hours: 20+ hours

Define your mini-project

AI-based character recognition for the Geʿez script, a more complex script

Our project is about creating and comparing different models for character recognition on a more complex non-Latin script, and we will thus also be working with image recognition which expands primarily on module three which was about AI tools, one of the topics covered there being just image recognition, accuracy and pre-processing.

We thought it would be interesting to further explore image recognition and since most handwritten character recognition models seem to be using the MNIST dataset, which is in the Latin script, we wanted to see how it would fare on a more complex script like the Geʿez abugida. The Geʿez script is used for languages like Amharic and Tigrinya as well as Geʿez.

We ended up implementing a support vector machine model and a convolutional neural network model. We did however not have enough time to implement more models nor implement them for the MNIST dataset which we would have done given more time. However, the main purpose was to implement and compare different approaches to image recognition of a more complex script.

Comparing the convolutional neural network and the support vector machine we can see that the support vector machine model performed slightly, but noticeably better than the CNN did on its validation set. This is interesting since SVM:s are generally supposed to be worse when there are much more labels, which there are with complex scripts. However, 238 different characters might not be that much, as to cause a problem for it.

It could be that the accuracy of the CNN could be improved by improving the network itself somehow or by more pre-processing of the data. Since the training accuracy continued to rise quite a bit above the validation accuracy it could be the case that it was overfitting and thus that could be something that could be looked into more given more time.

Working with this project we have learned a great deal about pre-processing as we ran into many a problem in the beginning regarding this. We also learned a lot more about image classification, support vector machines, and convolutional neural networks in depth.

If more time was at our disposal we would have liked to implement the model for a Latin-based dataset as well, like the MNIST dataset, so we could compare the same model's performance on a simpler versus a more complex one. But we can now conclude that the model works rather okay on complex scripts.

CNN model implementation

Introduction:
Note that the code is provided in the notebook which is also at the end of the report since the code for the CNN model is too large to fit in this section.

The code implements a convolutional neural network capable of correctly classifying the characters from the image. CNN:s are quite often used in image classification and they work similarly to how humans process images, that is to say by dividing it into smaller sub-pictures which it does in the convolutional layer.

The pooling layer works similarly by further shrinking the sub-images and then taking the max value in this sub-image (note that you can also choose to take the average instead of average-pooling). This is supposed to filter out unimportant noise from the image. The CNN consists of two convolutional layers followed by max-pooling layers in order to reduce the dimensionality further, which is important if the picture is large, although our dataset was not that large so two seemed to suffice.

The final layers are where the model learns to make the connections into the output and thus learns to classify the characters.

Data:
The data, which is split into two files, is first uploaded and then both files are unzipped to a folder whereby the images of characters are sorted into subfolders corresponding to their class, the character they represent, which can be read from the file name using the split() function. The size of the pictures in the dataset is 28x28.

We then utilize the created folder structure to create the test and validation sets using the function from Keras tf.keras.utils.image_dataset_from_directory() where we also apply grayscale. This function finds the images and interprets their classes by itself using the fact that they are stored in subfolders corresponding to their class. The validation-split is set to 0,2.

Furthermore, we normalize the data by dividing each value by 255, so that they go from ranging between 0 and 255 to ranging between 0 and 1. This actually gave a slight accuracy boost.

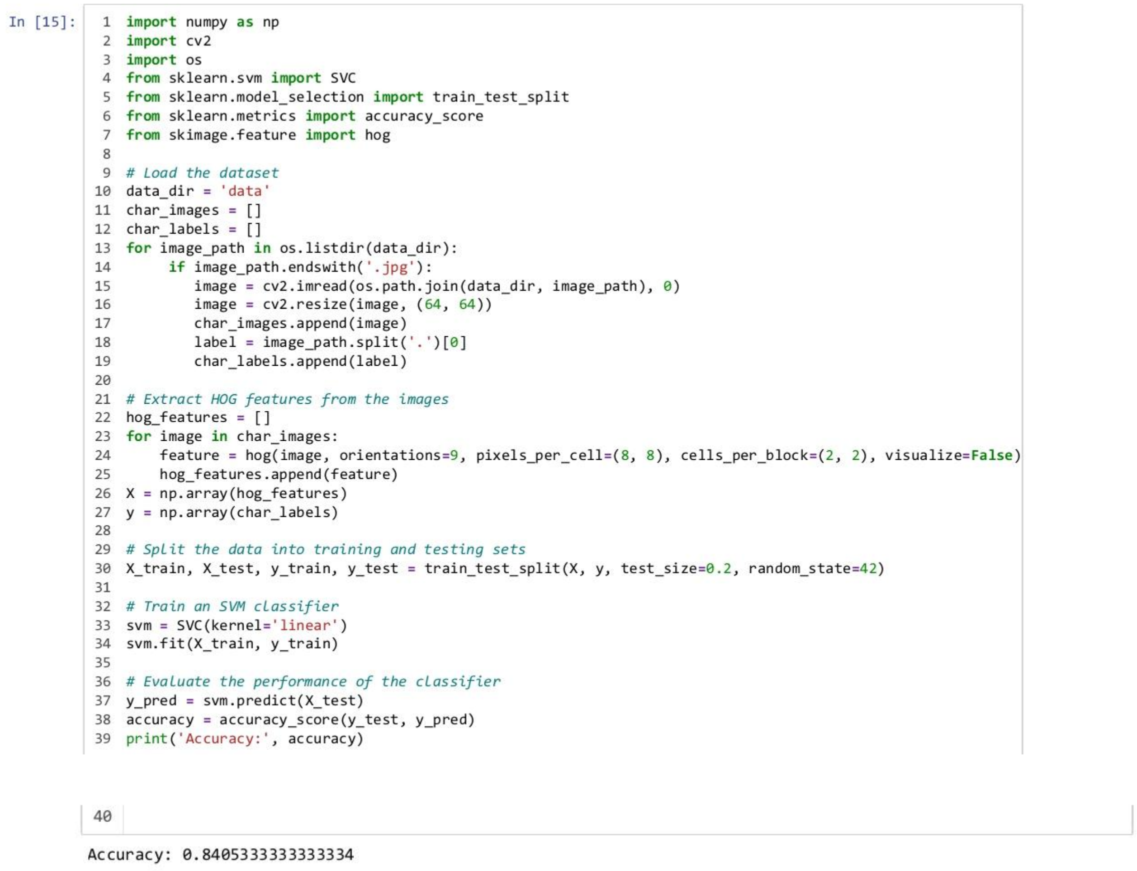
Model Training:
Using the training and validation sets we train our model which is a convolutional neural network consisting of various layers. The CNN is created by a function we made that takes into account the number of classes and image size, which is good if we would in the future extend the project for more datasets. The network has two Conv2D layers, the first one taking into account the image size, each one followed by a MaxPooling2D layer.

We then have a flattening layer, followed by two dense layers where the last one has units set to the number of classes. For the loss function, we used sparse categorical cross-entropy.

Model Evaluation and result:
The model was trained for 15 epochs with batch size set to 32. The accuracy rose very quickly at first, being around 0,75 at the second epoch. In the end, it had an accuracy of 0,98 on the training set and 0,79 on the validation set. As shown by the graph in the notebook, the validation accuracy quickly plateaued while the training accuracy continued rising quite a bit more.

Conclusion:
The code demonstrates how convolutional neural networks can be used for image recognition and that they work quite well for character recognition even on more complex scripts. The code was made so that it would be simple to extend it with different datasets, where you would only have to configure some of the preprocessing and give the right arguments to the function for creating the CNN.

SVM model implementation



```
In [15]:
import numpy as np
import cv2
import os
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from skimage.feature import hog

# Load the dataset
data_dir = 'data'
char_images = []
char_labels = []
for image_path in os.listdir(data_dir):
     if image_path.endswith('.jpg'):
        image = cv2.imread(os.path.join(data_dir, image_path), 0)
        image = cv2.resize(image, (64, 64))
        char_images.append(image)
        label = image_path.split('.')[0]
        char_labels.append(label)

# Extract HOG features from the images
hog_features = []
for image in char_images:
    feature = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=False)
    hog_features.append(feature)
X = np.array(hog_features)
y = np.array(char_labels)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train an SVM classifier
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)

# Evaluate the performance of the classifier
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8405333333333334
```

Introduction:
The provided code is a Python script that demonstrates the use of HOG (Histogram of Oriented Gradients) features and SVM (Support Vector Machine) classifier to classify characters. The code reads character images from a directory and extracts their HOG features to build a classifier that can predict the character in a new image.

HOG (Histogram of Oriented Gradients) features are often used for object recognition and image classification tasks because they capture information about the shape and texture of objects in an image. In the case of handwritten character recognition, HOG features can be used to extract information about the shape and texture of individual characters, which can be used to distinguish one character from another.

Data:
The code reads character images from the 'data' directory. The images are in JPG format and are resized to 64x64 pixels using OpenCV's cv2.resize() function. The images are then stored in a list along with their corresponding labels, which are extracted from the image file names using the split() method.

Feature extraction:
The HOG features are extracted from the images using the skimage.feature.hog() function. The parameters used for feature extraction are orientations=9, pixels_per_cell=(8, 8), and cells_per_block=(2, 2). These parameters define the number of orientation bins, the size of the cells, and the number of cells per block. The extracted features are stored in a list called hog_features.

Model Training:
The data is split into training and testing sets using the train_test_split() function from scikit-learn. The testing set size is set to 0.2, which means that 20% of the data is used for testing and the remaining 80% is used for training. The random_state parameter is set to 42 to ensure reproducibility of the results.

An SVM classifier is then trained on the training data using the SVC() function from scikit-learn. The kernel used for SVM is linear.

Model Evaluation:
The trained classifier is used to predict the labels of the test data using the predict() method of the SVM classifier. The predicted labels are compared with the actual labels of the test data, and the accuracy score is calculated using the accuracy_score() function from scikit-learn.

Result:
The output of the script is the accuracy score of the trained classifier, which is printed using the print() function. The obtained accuracy score is 0.8405333333333334.

Conclusion:
The provided code demonstrates the use of HOG features and SVM classifier to classify characters. The accuracy obtained is reasonably good, indicating that the classifier is effective in classifying the characters. The code can be extended to classify other types of images by modifying the feature extraction method and the classifier.

Comparison of the Approaches
We can compare the performance of the two approaches by looking at the accuracy scores achieved on the testing set. Running the above code, we get an accuracy of 0.84 for the SVM and an accuracy of 0.98 for the CNN. This shows that the CNN significantly outperforms the SVM in this task. This is not surprising, as CNNs are well-suited for image classification tasks, especially when there is a large amount of data available for training.

Reflection on the course as a whole [Group]

Through the course, we have covered a variety of topics including machine learning, natural language processing, dialogue systems, reinforcement learning, and deep learning. We have also explored the applications of AI in different fields such as healthcare, finance, and gaming.

From all the given coursework, we learned how to implement basic AI systems using Python, such as a weather forecast or a digital assistant. We also covered the importance of data and how to preprocess it for use in machine-learning models. Additionally, we learned about different types of models such as decision trees, random forests, and neural networks, and how to evaluate their performance as performance evaluation is not just as simple as looking at the accuracy but rather many factors that play into the accuracy of a model.

An interesting topic was that of interpretability and how it falls on a spectrum rather than being just black and white, interpretable or not. The paper we read about this topic in assignment five was very interesting and they mentioned that many authors don’t clearly define how they determine the interpretability in their model, which is very important to define.

We also learned that there is a balance one has to consider when developing systems between the accuracy and interpretability and that many times it might be better to have a system where actions it takes can be understood even if it means a lower accuracy. And this leads us into ethical questions as well regarding, for example, diagnostic systems.

Furthermore, the discussions have emphasized the importance of ethical considerations in AI, such as bias in data and models, and the need for transparency and accountability in AI decision-making. We have also explored the potential benefits and drawbacks of AI, and how it can be used to improve our lives while also being mindful of its limitations and potential risks.

Overall, the course was very interesting and we covered a wide variety of different topics in the field of artificial intelligence and machine learning and the design of systems based on them. We do however think that it could have been good to have a bit more practical parts in the lectures, maybe going over the basics of how to work with and create models for the module at hand, since this would very much help with the implementation part of the assignments which we think was not really focused on during the lectures and many students had difficulties with it.

Link to the dataset used

GitHub - Fetulhak/Handwritten-Amharic-character-Dataset