SREE NANDHA SIVAKUMARAN
QX54175

CAPTCHA Recognition Using Deep Learning

Abstract:

CAPTCHA recognition is a very challenging task in the field of computer vision. This project aims to develop a deep learning-based solution using Convolutional Neural Networks (CNNs) that decodes CAPTCHA images into their respective alphanumeric sequences. The model leverages deep-learning techniques, PyTorch, and GPU acceleration in an efficient training and testing process. The write up spells out the approach, implementation, and results, leaning on Edward Raff's Inside Deep Learning for the theoretical background and referring to outside sources for extra code and concepts.

Introduction:

CAPTCHA recognition is a very interesting task in the computer vision field. The goal of this project is to design a deep learning solution based on Convolutional Neural Networks (CNNs) that can convert CAPTCHA images into their corresponding sequences of alphanumerical characters. The model, therefore, employs deep learning techniques, PyTorch, and GPU. CAPTCHA is a common security system used to tell the difference between human users and automated bots. These tests usually contain distorted images with letters and numbers that users need to recognize. While effective against bots, the deliberate distortions, varying fonts, overlapping characters, and added noise of CAPTCHAs present problems for automated systems in solving them.

This task is to construct a deep learning solution for CAPTCHA recognition. The proposed model is trained with CNNs in order to transform CAPTCHA images into their corresponding sequences of letters and numbers. The approach integrates data preparation, CNN design, and training techniques in dealing with the challenges of CAPTCHA data.

Objective:

The main purpose of this project is to build a deep learning model that could correctly solve CAPTCHA pictures. This goal is divided into the following key components:

1. **Dataset Preparation:**
    - Pre-processing CAPTCHA images to ensure uniformity in size and format.
    - Try data augmentation techniques to help the model generalize better to new data.
2. **Model Development:**
    - Constructing a CNN architecture that efficiently extracts features from the images and maps them to multiple labels.
3. **Training and Optimization:**
    - The optimization will be improved by using state-of-the-art optimization methods: adaptive optimizers and learning rate planners.
    - Utilized data loading and augmentation techniques for efficient and robust training.
4. **Evaluation:**
    - Evaluate the performance and reliability of the model in recognizing unseen CAPTCHA images.
    - Analysing Prediction Errors to Find Areas to Improve.

SREE NANDHA SIVAKUMARAN
QX54175

This project shows how deep learning is applied to solve CAPTCHA recognition problems and also its ability to be applied to all other automatic visual recognition applications.

Dataset and Preparation:

In the project at hand, there was a dataset of CAPTCHA images, each containing an alphanumeric sequence of variable length. Besides this, the design of the CAPTCHA images contains distortions, overlapping characters, noise, and random variations in font and *colour* to make them difficult for automated systems. Each image in this problem corresponds to a sequence of labels, so special *pre-processing* and encoding will be carried out. In this respect, dataset preparation has entailed several critical steps in assuring its homogeneity along with its robustness for deep learning.

Steps in Dataset Preparation

1. **Loading and Label Extraction:**

    - CAPTCHA images and their corresponding labels are stored in a directory.
    - Labels embedded in the filenames are programmatically extracted to create the ground truth.

2. **Resizing:**

    - All images are resized to a standard shape, such as 128x128 pixels, to ensure uniform input dimensions, as neural networks require fixed-sized inputs.

3. **Normalization:**

    - Pixel values are normalized to a range of [0, 1] or standardized using the dataset's mean and standard deviation. This helps maintain stable gradient calculations during training.

4. **Data Augmentation:**

    - The model is made more robust to variations using augmentation techniques like random rotations, color jittering, Gaussian noise, and flipping.
    - These augmentations simulate real-world distortions and expand the effective dataset size.

5. **Label Encoding:**

    - Alphanumeric characters are mapped to integers using a dictionary (e.g., 'a': 0, 'b': 1, ..., '9': 35).
    - The entire sequence in each image is encoded as a list of integers, forming the ground truth for training.

6. **Dataset Splitting:**

    - The dataset is divided into training and validation sets, typically with an 80:20 ratio, to monitor generalization and avoid overfitting.

7. **Batching and Loading:**

    - Using PyTorch's DataLoader, the dataset is divided into mini-batches for efficient training. This ensures optimal memory usage and accelerates computations, especially on GPUs.

This is properly prepared, hence a consistent, variant, and deep learning-suitable dataset. Resizing and normalization put the inputs into standard size and further enhance their generalization capability. Further, splitting and batching of such a dataset will allow efficient model training and evaluation, making scenarios visualized by the model highly variant yet computationally efficient. These steps together lay the pipeline for successful CAPTCHA recognition.

Model Overview and Architecture

The CAPTCHA recognition model is built using a Convolutional Neural Network (CNN)**,** a type of deep learning model specifically designed for image processing tasks. The architecture is tailored to handle the challenges posed by CAPTCHA images, such as overlapping characters, noise, and distortions. The model performs multi-label classification, predicting sequences of alphanumeric characters from the input images.

Model Architecture

The architecture of the CNN includes the following components:

1. **Convolutional Layers:**
    - Extract low-level features (e.g., edges, textures) and progressively higher-level features (e.g., shapes and patterns) from the image.
    - Employ kernels (filters) that slide across the image to detect specific features.
    - Use **ReLU** activation to introduce non-linearity, enabling the network to learn complex patterns.

2. **Pooling Layers:**
    - Reduce the spatial dimensions of feature maps, retaining the most important information while discarding noise.
    - Max Pooling is used to capture the most prominent features, making the model robust to slight translations or distortions.

3. **Flattening:**
    - Converts the 2D feature maps into a 1D vector to feed into the fully connected layers.

4. **Fully Connected Layers:**
    - Perform classification by mapping the extracted features to output predictions.

- The final layer outputs probabilities for each possible character at every position in the CAPTCHA sequence.

5. **Softmax Activation:**
   - Ensures that the output probabilities for each position in the sequence sum to 1.
   - Helps in interpreting the output as a probability distribution over possible character classes.
   -

Working of the Model

1. **Input:**
   The model accepts a CAPTCHA image resized to a fixed dimension of 128x128 pixels as input. Before feeding the image into the model, it undergoes pre-processing steps like resizing, normalization, and augmentation to ensure uniformity and robustness.

2. **Feature Extraction:**
   Convolutional layers are used to extract features in a hierarchical manner. Initially, simple patterns like edges are identified, which later evolve into recognition of more complex shapes and structures within the image.

3. **Spatial Reduction:**
   Pooling layers reduce the dimensionality of feature maps, helping the model focus on the most significant patterns while also improving computational efficiency.

4. **Flattening and Classification:**
   The feature maps are flattened and passed through fully connected layers, which map the extracted features to probabilities for each character in the CAPTCHA sequence.

5. **Prediction:**
   The model predicts a probability distribution for each character in the sequence. The character with the highest probability at each position is selected as the predicted label.

This CNN-based architecture strikes a balance between accuracy and computational efficiency, making it an ideal choice for solving CAPTCHA recognition problems.

Training Process and Preparations

The training process of the CAPTCHA recognition model basically deals with optimizing its parameters, trying to minimize the difference between predicted labels and actual labels. This is done by a structured approach involving data preparation, an optimizer, loss function, and metrics. The goal is to make sure that the model learns profoundly from the training dataset, without forgetting generalization on CAPTCHA image data it has never seen before.

Data Pre-processing

- **Resizing:** All images are resized to a fixed dimension (128x128 pixels) to ensure uniformity.
- **Normalization:** Pixel values are normalized to the range [0, 1], stabilizing the gradient descent process during training.
  Data Augmentation: Random rotations, color jittering, and noise are applied to improve the model's robustness against variations in the dataset.

Dataset Splitting

- The dataset is divided into training and validation sets, typically using an 80-20 split. This ensures that the model is evaluated on unseen data to monitor its generalization.

Batch Loading

- The training and validation data are loaded into mini-batches using PyTorch's DataLoader, which allows efficient memory usage and parallel processing.

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

Loss Function

- CrossEntropyLoss is used for each position in the CAPTCHA sequence. It calculates the loss as the difference between the predicted probability distribution and the actual label.

Optimizer

- Adam Optimizer: Chosen for its adaptive learning rate capabilities, making it well-suited for noisy gradients.

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

Training Process

- Each batch of input images is passed through the CNN, generating a sequence of character probabilities for each image.
- The predicted outputs are compared to the true labels using the CrossEntropyLoss function. This loss quantifies the error in the predictions.
- Gradients of the loss with respect to model parameters are computed using automatic differentiation in PyTorch.
- The optimizer updates the model parameters based on the computed gradients to minimize the loss.
- After each epoch, the model is evaluated on the validation set. Metrics such as validation accuracy and loss are tracked to monitor the model's performance.

```
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
```

```
        outputs = model(images)
        loss = loss_function(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss / len(train_loader)}")
```

- The best-performing model (based on validation loss) is saved during training to prevent overfitting and ensure the availability of the most effective weights.
- The learning rate scheduler reduces the learning rate after a specified number of epochs to ensure convergence.

It guarantees that the model generalizes well to previously unseen CAPTCHA images and effectively picks up knowledge. Implementation of sophisticated optimizers, various regularization methods, and a tidy way of training avoids overfitting and makes the model perform on a completely new validation dataset with an accuracy which could not be attained earlier. All this lays down a base that assists in performing effective deep-learning driven recognition of CAPTCHAs.

Results

This further shows that the training is quite uniform in its learning process; hence, these epochs manage to smoothly and gradually reduce the loss values. Loss starts out relatively low to show that the initial architecture or data pre-processing pipeline was quite decent. At the end of training, the loss converges at approximately 0.0735, meaning that the error function has been well minimized. The highest training accuracy reached was 95.3%, proving that this model guessed most CAPTCHA sequences correctly in this training dataset. It is the fact that such differing accuracies become especially pronounced in later epochs that hints at the presence of noise-or equivalently, batch variability-strongly affecting the learning process.

The overall performance of the train demonstrates that, intrinsically, it identifies the core pattern within the CAPTCHA image through the CNN itself, which comprises character shapes and spacing to make the effective prediction, which can be taken further with increased epochs in order that these should develop better stability.

Conclusion

This project presents a strong deep learning-based pipeline for the recognition of CAPTCHAs. It gathers careful dataset preparation, a well-planned CNN architecture, and an effective training process that overcomes complications in CAPTCHA images. The model achieved very high accuracy on the validation set, hence very good generalization over unseen data. Future improvements, based on attention mechanisms and transfer learning, for example, might raise the efficiency of the system for each type of CAPTCHA. This research tends to prove the

capability of deep learning in automating complex visual tasks, like recognition of CAPTCHAs.

References:

1. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems.
2. Noury, Z., & Rezaei, M. (2020). Deep-CAPTCHA: A Deep Learning-Based CAPTCHA Solver for Vulnerability Assessment. Faculty of Computer and Electrical Engineering, Qazvin Azad University; Institute for Transport Studies, University of Leedsllow, I., et al. (2016). Deep Learning. MIT Press.
3. Von Ahn, L., et al. (2003). CAPTCHA: Using Hard AI Problems for Security. EUROCRYPT.
4. He, K., et al. (2016). Deep Residual Learning for Image Recognition. CVPR Proceedings.
5. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.