ML_Week14_Report

NAME : CHETAN NADICHAGI
SRN : PES2UG23CS149
SEC : C

1. Introduction:-

The objective of this lab was to design, train, and evaluate a Convolutional Neural Network (CNN) to classify images of hand gestures representing the game Rock, Paper, Scissors using a labelled image dataset. The task covered the complete deep learning pipeline, including dataset preparation, CNN architecture design, model training with backpropagation, and performance evaluation on a held-out test set.

2. Model Architecture:-

The implemented CNN, named RPS_CNN, takes RGB images of size 128 × 128 as input and processes them through a stack of three convolutional blocks followed by a fully connected classifier. Each convolutional block uses a kernel size of 3 × 3 with padding = 1 to preserve spatial resolution before down sampling, and applies nonlinearity with ReLU activations.

- The **convolutional block** (conv_block) is defined as:
  - First conv layer: Conv2d(3, 16, kernel_size=3, padding=1) → ReLU() → MaxPool2d(2); this maps the 3-channel input to 16 feature maps while halving the spatial dimensions from 128 × 128 to 64 × 64.

- Second conv layer: Conv2d(16, 32, kernel_size=3, padding=1) → ReLU() → MaxPool2d(2); this increases channels to 32 and reduces spatial size from 64 × 64 to 32 × 32.
- Third conv layer: Conv2d(32, 64, kernel_size=3, padding=1) → ReLU()
  → MaxPool2d(2); this further increases channels to 64 and reduces spatial size from 32 × 32 to 16 × 16.

Max pooling with kernel size 2 × 2 and stride 2 is used after each convolutional layer to progressively downsample the feature maps, reducing computation and providing translational invariance. After three pooling operations, an original 128 × 128 image becomes a 16 × 16 feature map with 64 channels, giving a flattened dimension of 64 × 16 × 16 = 16384 features.

The **fully connected classifier** (fc) is implemented as:

- Flatten() to convert the 64 × 16 × 16 tensor into a 1D vector per image. oLinear(64 * 16 * 16, 256) followed by ReLU() to project the highdimensional feature vector into a 256-dimensional latent representation with non-linear activation.
- Dropout(p=0.3) to regularize the model and reduce overfitting by randomly dropping 30% of the neurons during training.
- Final Linear(256, 3) layer to produce logits corresponding to the three classes: rock, paper, and scissors.

3. Training and Performance:-

The dataset is first downloaded from Kaggle and organized into classspecific folders (rock, paper, scissors), then loaded using torchvision.datasets .ImageFolder from the directory

/content/dataset. All images are preprocessed by resizing to 128 × 128, converting to tensors, and normalizing each RGB channel with mean [0.5,0.5,0.5] and standard deviation [0.5,0.5,0.5].

The full dataset consists of 2188 images, which are split into 80% training and 20% testing: 1750 images for training and 438 images for testing. Data loaders are created with a batch size of 32, using shufling for the training loader and no shufling for the test loader to ensure reproducible evaluation.

The key **hyperparameters and training setup** are:

- Device: CPU (as reported by print("Using device:", device) when CUDA is not available).
- Loss function: CrossEntropyLoss, which is standard for multi-class classification problems and matches the final logits output of the network.
- Optimizer: Adam optimizer (optim.Adam) with a learning rate of 0.001, providing adaptive learning rates per parameter and typically faster convergence than vanilla SGD.
- Number of epochs: 10 full passes over the training data (EPOCHS = 10).

During training, for each batch, the gradients are reset using optimizer.zero_grad(), a forward pass is executed (outputs = model(images)), and the loss is computed as criterion(outputs, labels). Backpropagation is performed with loss.backward(), and the model parameters are updated via optimizer.step(), while the average epoch loss is printed for monitoring convergence.

The **training loss** decreases consistently across epochs, with the epoch-wise losses approximately: 0.7199, 0.2139, 0.1191, 0.0518, 0.0411, 0.0303, 0.0338, 0.0161, 0.0092, and 0.0094 for epochs 1 through 10 respectively, indicating good optimization progress and low final training error. After training, the model is evaluated on the test set using torch.no_grad() and model.eval(), computing the number of correct predictions compared to the labels.

The final **Test Accuracy** achieved by the model on the unseen test set is reported as:

- Test Accuracy: 99.32%

Additionally, a helper function predict_image demonstrates singleimage inference; for example, the model correctly predicts the label paper for the image /content/dataset/paper/0Uomd0HvOB33m47I.png. A simple Rock–Paper–Scissors “game” is then implemented where two random images are selected, classified by the model, and the winner is decided based on the standard game rules, illustrating that the trained CNN can reliably drive an interactive application.

4. Conclusion and Analysis:-

The constructed CNN model performs very well on the Rock–Paper– Scissors image classification task, achieving a high test accuracy of 99.32% on the held-out test set. The consistently decreasing training loss and strong generalization to the test data suggest that the chosen architecture and hyperparameters are appropriate for this dataset.

One major factor contributing to the strong performance is the multistage convolutional feature extractor, which gradually increases the number of channels from 3 to 64 while reducing spatial resolution, allowing the network to learn both low-level edges and higher-level shape patterns characteristic of hand gestures. The use of ReLU activations and Max Pooling after each convolutional layer helps in capturing translation-invariant features and stabilizes training. The fully connected block with 256 hidden units, combined with dropout, provides sufficient capacity to map features to the three output classes while mitigating overfitting.

Some challenges in such a lab may include tuning hyperparameters like learning rate, number of epochs, and batch size to ensure stable convergence without overfitting, especially when training on CPU where experimentation is slower. Ensuring correct preprocessing— such as consistent resizing, normalization, and proper train-test splitting—is also crucial; misconfiguration at this stage can significantly degrade performance or cause label mismatches.

There are a few potential improvements that could be explored to possibly increase robustness and accuracy further:

- Introduce data augmentation (random rotations, horizontal flips, slight translations, and brightness/contrast adjustments) to make the model more invariant to variations in hand orientation and lighting conditions, which may yield better generalization on more diverse real-world images.

- Experiment with a slightly deeper architecture or adding batch normalization layers after convolutional layers to stabilize training and allow for higher learning rates, which might improve convergence and robustness to initialization.

- Perform hyperparameter tuning on learning rate, number of epochs, and dropout rate (e.g., 0.4 or 0.5) to balance bias and variance, as well as experimenting with early stopping based on validation performance rather than a fixed epoch count.