IIIT BANGALORE


Visual Recognition

AI825

# Assignment 3

*Authors:*
Karan Pandey
IMT2021014
Anuj Arora
IMT2021050
Prasad Jore
IMT2021104

March 22, 2024

# 1 A

## Playing around with a medium deep network similar to AlexNet.

**The Dataset:**
We are using the CIFAR-10 dataset. It is a benchmark dataset used for image classification. It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The classes include common objects such as airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. CIFAR-10 serves as a standard dataset for evaluating and comparing the performance of machine learning algorithms and models in image classification tasks.
**About AlexNet:**
AlexNet is a convolutional neural network (CNN) architecture designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It consists of eight layers, including five convolutional layers followed by three fully connected layers. lexNet was groundbreaking for its time due to its deep architecture and the use of techniques such as ReLU activation functions, dropout regularization, and data augmentation.
**Input Images:**
In the CIFAR-10 dataset, the size of the images is 32x32 pixels. These images are RGB (3 channels), meaning they have a height of 32 pixels, a width of 32 pixels, and three color channels (red, green, and blue).
**Our Model:**
NoteBook on GitHub
**Convolutional Layers:**

- Conv1: Takes input images with three channels (RGB) and applies 64 filters of size 5x5 with padding of 2 and ReLU activation.
- MaxPool1: Performs max-pooling with a kernel size of 3x3 and stride of 2 after Conv1.
- Conv2: Applies 192 filters of size 5x5 with padding of 2 and ReLU activation.
- MaxPool2: Performs max-pooling with a kernel size of 3x3 and stride of 2 after Conv2.

- Conv3: Applies 384 filters of size 3x3 with padding of 1 and ReLU activation.
- MaxPool3: Performs max-pooling with a kernel size of 3x3 and stride of 2 after Conv3.

**Fully Connected Layers (FC):**

- FC1: Consists of 512 neurons followed by ReLU activation.
- FC2: Consists of 256 neurons followed by ReLU activation.
- FC3: Consists of 128 neurons followed by ReLU activation.

**Output Layer:**

- The output layer consists of num_classes neurons corresponding to the number of classes in the classification task.

**Activation Function:**

- ReLU (Rectified Linear Unit) activation function is used after each convolutional layer and fully connected layer except for the output layer.
- ReLU introduces non-linearity, allowing the model to learn complex patterns in the data.

**Pooling Layers:**

- Max-pooling layers are applied after each convolutional layer to down-sample the feature maps and reduce spatial dimensions.

**Optimizer (Adam):**

- The Adam optimizer is used to update the weights of the model during training.
- Adam stands for Adaptive Moment Estimation. It is an adaptive learning rate optimization algorithm that computes adaptive learning rates for each parameter by keeping track of both first and second moments of the gradients.

| Algorithm | ReLu | tanh | Sigmoid |
|---|---|---|---|
| Adam | 8640.88 | 2843.69 | 3962.27 |
| SGD without momentum | 7123.45 | 2691.12 | 3771.83 |
| SGD with momentum | 6912.52 | 2227.82 | 3512.16 |

Table 1: ReLu vs tanh vs Sigmoid

| Algorithm | ReLu | tanh | Sigmoid |
|---|---|---|---|
| Adam | 76.21% | 65.6% | 10.0% |
| SGD without momentum | 14.49% | 19.38% | 10.0% |
| SGD with momentum | 39.74% | 41.72% | 10.0% |

Table 2: Accuracy Comparsion

**Training and Validation** For the first test we used ADAM optimization algorithm and for the next run we will be using SGD(Stochastic gradient descent) with and without momentum. Table 1 gives a comparison of these training methods based on time taken(in seconds) to train.

**Accuracy:**
Table 2 gives a rough comparison of the accuracy of different models we have used in this test.

**Recommended Architecture:**
ADAM optimization algorithm outperforms SGD (with or without momentum) in terms of accuracy, albeit at the cost of longer training times. Its superior classification performance makes up for this drawback. Therefore, ADAM is the preferred optimization algorithm. When using ADAM, ReLU activation function yields higher accuracy compared to tanh and sigmoid, albeit with longer training times. In conclusion, it is recommended to use ADAM optimization algorithm with ReLU activation function for optimal results.

# 2 B

## CNN as a feature extracter

**The Dataset:** We use the Intel Image Classification dataset. This is image data of Natural Scenes around the world. This Data contains around 25k images of size 150x150 distributed under 6 categories. These are buildings, forest, glacier, mountain, sea, street.

**The Model:**
We will be using ALexNet model from the torchvision.models . This is pre trained model whose last layer will be altered without altering the weights of the pre-trained model.

```
# Modify the last layer to output features
# Get the number of input features to the last fully connected layer
num_features = alexnet.classifier[6].in_features

# Define the new linear layer with same output size as original last layer
desired_output_size = 1000 #output features
new_last_layer = torch.nn.Linear(num_features, desired_output_size)

# Replace the last layer with the new linear layer
alexnet.classifier[6] = new_last_layer
```

The final layer of the model is substituted with a Linear layer, which preserves the features of the input image without modifying the pre-trained model's weights. These features are then computed for all training and test data. Subsequently, a Logistic Regression classifier is trained on the training data using these computed features.
**Accuracy:**
For the Intel Image Classification data set we get accuracy score of 67.3%. In the case of the Bikes and Horses dataset, the best results were achieved using Logistic Regression, yielding an accuracy score of 98.3%.

# 3 C

## YOLO V2

YOLOv2, short for "You Only Look Once version 2," is an object detection algorithm introduced as an improvement over the original YOLO (You Only Look Once) algorithm. It was developed by Joseph Redmon and Ali Farhadi in 2016.
Salient features of YOLOv2 are:

- **High-Resolution Classifier:** In YOLOv2, Redmon and Farhadi perform the pretraining classification step with 224 × 224. Still, they fine-tune the classification network at the upscaled 448 × 448 resolution for ten epochs on the same ImageNet data. By doing this, the network got time. It adapted its filters to work better on the upscaled resolution since it had already seen that resolution in the fine-tuning classification step.

- **Convolution with Anchor Boxes:** YOLOv2 removes the fully connected layers and uses anchor boxes to predict bounding boxes. Hence, making it fully convolutional.Anchor boxes are a set of predefined boxes with a specific height and width; they act as a guess or prior for the objects in the dataset that help optimize the network fast. They are multiple bounding boxes (priors) with different aspect ratios and sizes centered on each pixel. The goal of an object detection network is to predict a bounding box and its class label.

- **Dimension Clusters:** dimension clusters refer to the process of clustering the sizes of ground truth bounding boxes to determine appropriate anchor box priors for the model. This clustering helps in initializing anchor boxes that better represent the distribution of object sizes in the dataset, leading to improved detection performance.The dimension clustering process involves grouping these bounding box dimensions into clusters based on similarity. This is typically done using techniques like k-means clustering.

- **Direct Location Prediction:** Direct location prediction revolutionized object detection by eliminating the need for anchor boxes. Instead,

it divides the input image into a grid and directly predicts bounding box coordinates relative to grid cells. This approach simplifies the prediction process and allows for more flexible handling of objects of different sizes and aspect ratios. YOLOv2's output tensor includes predictions for multiple bounding boxes per grid cell, along with class probabilities. By optimizing both localization and classification losses during training, YOLOv2 learns to accurately detect objects while minimizing false positives. This efficient and accurate detection method, coupled with non-maximum suppression for post-processing, makes YOLOv2 a popular choice for real-time object detection tasks.

- **Mutli Scale Training:** Multi-scale training is a technique used in object detection models like YOLOv2 to improve their robustness to objects of varying sizes in images. This approach involves training the model on images of different scales or resolutions, thereby enabling it to detect objects at different levels of granularity. During training, batches of images are randomly resized to different scales, and the model is trained on these varying scales. This helps the model learn to detect objects regardless of their size in the input image. Multi-scale training encourages the model to capture both fine-grained details and coarse structures, leading to improved performance on objects of different sizes during inference. It also helps prevent overfitting by exposing the model to a diverse range of input sizes. Overall, multi-scale training is a valuable technique for enhancing the versatility and accuracy of object detection models like YOLOv2.

# 4 D

## Object tracker: SORT + DeepSORT

**Implementation 1: Faster RCNN + SORT**

**Performance:** This implementation utilizes Faster RCNN for object detection and SORT for tracking.

**Observations:**

- **Object detection:** Faster RCNN provides accurate bounding box predictions for vehicles.

- **Tracking:** SORT effectively tracks vehicles but may suffer when multiple vehicles occlude each other.

**Results:** The tracker performs reasonably well, but occasional failures in tracking can occur due to occlusions.

**Implementation 2: Faster RCNN + DeepSORT**

**Performance:** This implementation also uses Faster RCNN for detection but employs DeepSORT for tracking.

**Observations:**

- **Object detection:** Similar to the previous implementation, Faster RCNN provides accurate bounding box predictions.
- **Tracking:** DeepSORT improves tracking accuracy, especially in scenarios with occlusions or crowded scenes.

**Results:** The tracker demonstrates improved robustness compared to SORT, particularly in handling occlusions and maintaining track identity.

**Implementation 3: YOLO v5 + SORT**

**Performance:** This implementation adopts YOLO v5 for object detection and SORT for tracking.

**Observations:**

- **Object detection:** YOLO v5 offers fast and efficient detection of vehicles with slightly different detection characteristics compared to Faster RCNN.
- **Tracking:** SORT effectively tracks vehicles but may face challenges in maintaining track identity across frames, especially in crowded scenes.

**Results:** The tracker performs adequately, but occasional identity switches or tracking failures may occur, especially in complex scenarios.

**Implementation 4: YOLO v5 + DeepSORT**

**Performance:** This implementation combines YOLO v5 for detection with DeepSORT for tracking. **Observations:**

- **Object detection:** YOLO v5 offers fast detection with slightly different detection characteristics compared to Faster RCNN.
- **Tracking:** DeepSORT provides robust and accurate tracking, even in challenging scenarios with occlusions and crowded scenes.

**Results:** The tracker exhibits superior performance compared to SORT, demonstrating improved accuracy and robustness, especially in complex scenarios.

**Difference between SORT and DeepSORT:**
**SORT (Simple Online and Realtime Tracking):**

- SORT is a simple yet effective tracking algorithm based on the Kalman filter and Hungarian algorithm.
- It predicts the state of each object in the next frame and assigns detections to existing tracks using a cost function.
- SORT does not include appearance features or deep learning-based methods for track association, making it susceptible to identity switches and occlusions in crowded scenes.

**DeepSORT (Deep Learning-based SORT):**

- DeepSORT extends SORT by incorporating deep appearance features for track association.
- It utilizes a deep neural network to extract appearance features from object detections, which are then used in conjunction with the motion model for track association.
- DeepSORT is more robust to occlusions and track identity switches, resulting in improved tracking performance, especially in complex scenes.

**Ground Truth Creation for Evaluation:** If we count the number of cars manually in the video, in most the cases, it is approximately same as the result given by the model.
Video results on github with notebook