VR Mini Project Report

P. Ramsai Koushik (IMT2021072)
Nilay Kamat (IMT2021096)
Barath.S.Narayan(IMT2021524)

March 22, 2024

Contents

1 Task A(Image Recognition): Training and Evaluating CNN based Models on CIFAR-10

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. It contains the following set of object categories: automobile, bird, cat, deer, dog, frog, horse, ship, truck.

In compliance with the assignment's instructions to explore medium deep networks, two different architectures were explored:

1. Architecture 1: This configuration consists of 2 convolutional layers followed by 2 fully connected layers.
2. Architecture 2: This configuration consists of 3 convolutional layers followed by 2 fully connected layers.

1.1 Architecture 1

1.1.1 Architecture 1 Overview

The CNN architecture comprises of the following components:

- **Input Layer:** The input consists of RGB images with three channels.
- **Convolutional Layers:** Two convolutional layers are utilized to extract features from the input images. Each convolutional layer is followed by batch normalization to stabilize and accelerate the training process.
- **Pooling Layers:** Max-pooling layers are employed to down-sample the feature maps obtained from the convolutional layers, preserving essential features while reducing spatial dimensions.
- **Flatten Layer:** Following the final pooling layer, a flatten layer reshapes the feature maps into a one-dimensional vector, facilitating the transition to the fully connected layers.
- **Fully Connected Layers:** Two fully connected layers are incorporated for classification purposes, leveraging the extracted features. Batch normalization is applied to the first fully connected layer for improved training stability.
- **Activation Function:** The chosen activation function is applied after each convolutional layer and the first fully connected layer to introduce non-linearity into the network.
- **Output Layer:** The output layer consists of a softmax activation function, producing class probabilities for multi-class classification tasks.

Layer Details

1. **Convolutional Layers:**
   - **Layer 1:**
     - Input Channels: 3 (RGB)
     - Output Channels: 64
     - Kernel Size: 3×3
     - Padding: 1
   - **Layer 2:**
     - Input Channels: 64
     - Output Channels: 128
     - Kernel Size: 3×3
     - Padding: 1

2. **Pooling Layers:** Max-pooling with a kernel size of 2×2 and a stride of 2 is applied after each convolutional layer.

3. **Fully Connected Layers:**

   - **Layer 1:**
     - Input Size: 8192 (computed based on output channels, height, and width after the convolutional and pooling layers)
     - Output Size: 1024
   - **Layer 2 (Output Layer):**
     - Input Size: 1024
     - Output Size: 10 (Number of classes)

4. **Activation Function:** The chosen activation function is applied after each convolutional layer and the first fully connected layer.

5. **Output Layer:** Softmax activation function is applied to produce class probabilities.

1.1.2 Common Hyperparameters

We kept the following hyperparameters same for all experiments to ensure an unbiased comparison of different activation functions, and different optimization strategies while training.

- **Epochs:** 20
- **Batch Size:** 64
- **Learning Rate:** 0.001

1.1.3 Experiments

1. **Experiment 1**: ReLU Activation Function without Momentum

   - **Activation Function and Optimizer:** ReLU activation function was utilized with the SGD optimizer without momentum.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot [1] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 10 seconds**.Test accuracy achieved: **61.28%**

2. **Experiment 2:** ReLU Activation Function with Momentum

   - **Activation Function and Optimizer:** ReLU activation function was utilized with the SGD optimizer, using momentum values of 0.5, 0.7, and 0.9.
   - **Results:**
     - **Momentum 0.5:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[1] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 49 seconds**. **Test accuracy achieved: 69.07%**.
     - **Momentum 0.7:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[1] of training accuracy over epochs for detailed visualization.The time taken to train the model was **5 minutes 58 seconds**. **Test accuracy achieved: 71.38%**.
     - **Momentum 0.9:**Training loss stabilized around a certain value over 20 epochs. Refer to the plot[1] of training accuracy over epochs for detailed visualization. The time taken to train the model was **6 minutes 16 seconds Test accuracy achieved: 74.17%**.

3. **Experiment 3**: ReLU Activation Function with Adam Optimizer

- **Activation Function and Optimizer:** ReLU activation function was utilized with the Adam optimizer.
- **Results:** Training loss decreased over 20 epochs. Refer to the plot[1] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 5 seconds**. **Test accuracy achieved: 76.87%**.

4. **Experiment 4:** Tanh without Momentum
   - **Activation Function and Optimizer:** Tanh activation function was utilized with the SGD optimizer without momentum.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[2] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 4 seconds**. **Test accuracy achieved: 57.44%**.
5. **Experiment 5:** Tanh Activation Function with Momentum
   - **Activation Function and Optimizer:** Tanh activation function was used with the SGD optimizer, employing momentum values of 0.5, 0.7, and 0.9.
   - **Results:**
     - **Momentum 0.5:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[2] of training accuracy over epochs for detailed visualization. The time taken to train the model was **6 minutes 5 seconds**. **Test accuracy achieved: 63.42%**.
     - **Momentum 0.7:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[2] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 46 seconds**. **Test accuracy achieved: 67.38%**.
     - **Momentum 0.9:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[2] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 49 seconds**. **Test accuracy achieved: 68.52%**.
6. **Experiment 6:** Tanh Activation Function with Adam Optimizer
   - **Activation Function and Optimizer:** Tanh activation function was utilized with the Adam optimizer.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[2] of training accuracy over epochs for detailed visualization. The time taken to train the model was **6 minutes 4 seconds**. **Test accuracy achieved: 72.16%**.
7. **Experiment 7:** Sigmoid without Momentum
   - **Activation Function and Optimizer:** Sigmoid activation function was utilized with the SGD optimizer without momentum.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[3] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 17 seconds**. **Test accuracy achieved: 40.04%**.
8. **Experiment 8:** Sigmoid Activation Function with Momentum
   - **Activation Function and Optimizer:** Sigmoid activation function was used with the SGD optimizer, employing momentum values of 0.5, 0.7, and 0.9.
   - **Results:**
     - **Momentum 0.5:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[3] of training accuracy over epochs for detailed visualization. The time taken to train the model was **6 minutes 9 seconds**. **Test accuracy achieved: 51.85%**.

- **Momentum 0.7:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[3] of training accuracy over epochs for detailed visualization.The time taken to train the model was **6 minutes 19 seconds**. **Test accuracy achieved: 57.01%**.
- **Momentum 0.9:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[3] of training accuracy over epochs for detailed visualization.The time taken to train the model was **6 minutes 26 seconds**. **Test accuracy achieved: 58.42%**.

9. **Experiment 9:** Sigmoid Activation Function with Adam Optimizer

   - **Activation Function and Optimizer:** Sigmoid activation function was utilized with the Adam optimizer.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[3] of training accuracy over epochs for detailed visualization.The time taken to train the model was **10 minutes 28 seconds**. **Test accuracy achieved: 68.89%**.

1.2 Architecture 2

1.2.1 Architecture 2 Overview

The CNN architecture consists of the following key components:

- **Input Layer:** The input images are typically RGB images with three channels.
- **Convolutional Layers:** Three convolutional layers are utilized to extract features from the input images. Each convolutional layer is followed by batch normalization to stabilize and accelerate the training process.
- **Pooling Layers:** Max-pooling layers are employed to down-sample the feature maps obtained from the convolutional layers, reducing their spatial dimensions while retaining important features.
- **Flatten Layer:** After the final pooling layer, a flatten layer reshapes the feature maps into a one-dimensional vector, preparing them for the fully connected layers.
- **Fully Connected Layers:** Two fully connected layers are incorporated to perform classification based on the extracted features. Batch normalization is applied to the first fully connected layer for improved training stability.
- **Activation Function:** The chosen activation function is used after each convolutional layer and the first fully connected layer to introduce non-linearity into the network.
- **Output Layer:** The output layer consists of a softmax activation function, producing class probabilities for multi-class classification tasks.

Layer Details

1. **Convolutional Layers:**

   - **Layer 1:**
     - Input Channels: 3 (RGB)
     - Output Channels: 64
     - Kernel Size: 3×3
     - Padding: 1
   - **Layer 2:**
     - Input Channels: 64
     - Output Channels: 128
     - Kernel Size: 3×3

- Padding: 1

- **Layer 3:**
  - Input Channels: 128
  - Output Channels: 256
  - Kernel Size: 3×3
  - Padding: 1

2. **Pooling Layers:** Max-pooling with a kernel size of 2×2 and a stride of 2 is applied after each convolutional layer.

3. **Fully Connected Layers:**

   - **Layer 1:**
     - Input Size: 4096
     - Output Size: 1024
   - **Layer 2 (Output Layer):**
     - Input Size: 1024
     - Output Size: 10 (Number of classes)

4. **Activation Function:**

   The chosen activation function is applied after each convolutional layer and the first fully connected layer.

5. **Output Layer:**

   Softmax activation function is applied to produce class probabilities.

1.2.2 Experiments

1. **Experiment 1**: ReLU Activation Function without Momentum

   - **Activation Function and Optimizer:** ReLU activation function was utilized with the SGD optimizer without momentum.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot [4] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 21 seconds**. Test accuracy achieved: **63.73%**

2. **Experiment 2:** ReLU Activation Function with Momentum

   - **Activation Function and Optimizer:** ReLU activation function was utilized with the SGD optimizer, using momentum values of 0.5, 0.7, and 0.9.
   - **Results:**
     - **Momentum 0.5:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[4] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 36 seconds**. **Test accuracy achieved: 70.38%**.
     - **Momentum 0.7:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[4] of training accuracy over epochs for detailed visualization.The time taken to train the model was **5 minutes 38 seconds**. **Test accuracy achieved: 73.79%**.
     - **Momentum 0.9:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[4] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 41 seconds**. **Test accuracy achieved: 75.71%**.

3. **Experiment 3**: ReLU Activation Function with Adam Optimizer

- **Activation Function and Optimizer:** ReLU activation function was utilized with the Adam optimizer.
- **Results:** Training loss decreased over 20 epochs. Refer to the plot[4] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 57 seconds**. **Test accuracy achieved: 80.21%**.

4. **Experiment 4:** Tanh without Momentum
   - **Activation Function and Optimizer:** Tanh activation function was utilized with the SGD optimizer without momentum.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[5] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 32 seconds**. **Test accuracy achieved: 58.61%**.
5. **Experiment 5:** Tanh Activation Function with Momentum
   - **Activation Function and Optimizer:** Tanh activation function was used with the SGD optimizer, employing momentum values of 0.5, 0.7, and 0.9.
   - **Results:**
     - **Momentum 0.5:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[5] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 41 seconds**. **Test accuracy achieved: 64.52%**.
     - **Momentum 0.7:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[5] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 40 seconds**.**Test accuracy achieved: 69.08%**.
     - **Momentum 0.9:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[5] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 41 seconds**.**Test accuracy achieved: 70.84%**.
6. **Experiment 6:** Tanh Activation Function with Adam Optimizer
   - **Activation Function and Optimizer:** Tanh activation function was utilized with the Adam optimizer.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[5] of training accuracy over epochs for detailed visualization. The time taken to train the model was **6 minutes 4 seconds**. **Test accuracy achieved: 73.37%**.
7. **Experiment 7:** Sigmoid without Momentum
   - **Activation Function and Optimizer:** Sigmoid activation function was utilized with the SGD optimizer without momentum.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[6] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 37 seconds**. **Test accuracy achieved: 43.79%**.
8. **Experiment 8:** Sigmoid Activation Function with Momentum
   - **Activation Function and Optimizer:** Sigmoid activation function was used with the SGD optimizer, employing momentum values of 0.5, 0.7, and 0.9.
   - **Results:**
     - **Momentum 0.5:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[6] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 44 seconds**. **Test accuracy achieved: 50.1%**.

- **Momentum 0.7:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[6] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 47 seconds**. **Test accuracy achieved: 58.83%**.
- **Momentum 0.9:** Training loss stabilized around a certain value over 20 epochs. Refer to the plot[6] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 47 seconds**.**Test accuracy achieved: 62.04%**.

9. **Experiment 9:** Sigmoid Activation Function with Adam Optimizer
   - **Activation Function and Optimizer:** Sigmoid activation function was utilized with the Adam optimizer.
   - **Results:** Training loss decreased over 20 epochs. Refer to the plot[6] of training accuracy over epochs for detailed visualization. The time taken to train the model was **5 minutes 54 seconds**. **Test accuracy achieved: 63.96%**.

Observations and Conclusion

Observations:

- Among the various activation functions and optimizers tested, the model with **ReLU activation function and Adam optimizer in the second architecture** achieved the highest test accuracy(**80.21%**).
- The training times for all models were approximately 5 minutes.

Conclusion:

- Based on the observations, the model utilizing **ReLU activation function with the Adam optimizer in the second architecture** stands out as the best performer.
- This model not only attained the highest accuracy but also demonstrated reasonable training times, making it the best choice for image classification.

2 Task B : Usage of CNNs as feature extractors

2.1 Dataset Used:

For this task, we have used the "Flowers102" dataset. Oxford 102 Flower is an image classification dataset consisting of 102 flower categories. The flowers were chosen to be flowers commonly occurring in the United Kingdom. Each class consists of between 40 and 258 images. The images have large scale, pose and light variations. In addition, there are categories that have large variations within the category, and several very similar categories.

2.2 Pre-Processing:

We have split the dataset into train, test, and validation datasets. This is done to ensure that overfitting can be detected in the training stage itself. We further performed the following pre-processing steps on images in both train and test data:

1. Resizing of Image to 224x224 to fit the AlexNet specifications.
2. Normalising RGB channels with mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225].

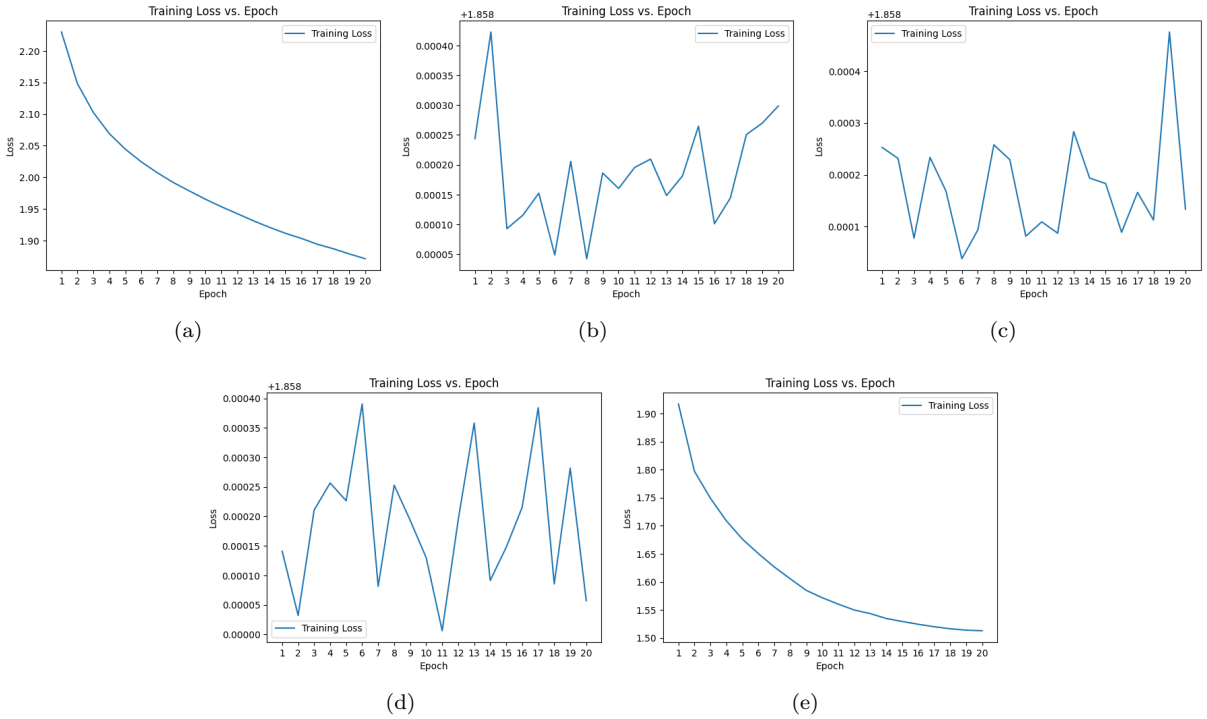

Figure 1: Snapshots showing Training Loss over Epochs for First Architecture with ReLU as activation function (a) Without Momentum (b) Momentum = 0.5 (c) Momentum = 0.7 (d) Momentum = 0.9 (e) With Adam

2.3 Results:

Food101 Dataset

1. **AlexNet Architecture**

   (a) We used the AlexNet model as a feature extractor using PyTorch's pre-trained model.

   (b) Classifiers used and their accuracy results obtained are as follows:

Classifiers Used	**Accuracy**
Logistic Regression	0.91320293398533
SVC(gamma='auto',kernel='linear')	0.902200488997555
RandomForestClassifier(max_depth=20, random_state=42)	0.6528117359413202
RandomForestClassifier(random_state=42)	0.6687041564792175
KNeighborsClassifier(n_neighbors=5)	0.589242053789731

Bikes Vs Horses Dataset

1. **AlexNet Architecture**

   (a) We used the AlexNet model as a feature extractor using PyTorch's pre-trained model.

   (b) Classifiers used and their accuracy results obtained on using all features are as follows:

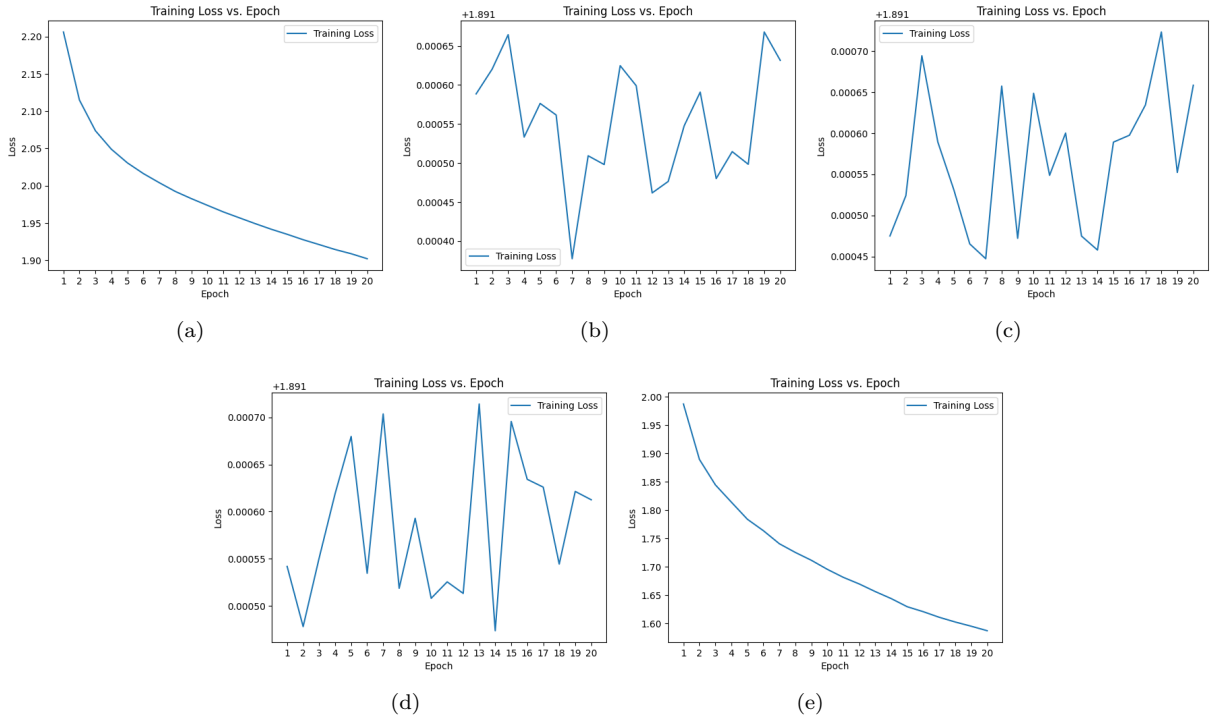

Figure 2: Snapshots showing Training Loss over Epochs for First Architecture with Tanh as activation function (a) Without Momentum (b) Momentum = 0.5 (c) Momentum = 0.7 (d) Momentum = 0.9 (e) With Adam

Classifiers Used	**Accuracy**
Logistic Regression	1.0
SVC(gamma='auto',kernel='linear')	1.0
RandomForestClassifier(max_depth=20, random_state=42)	1.0
RandomForestClassifier(random_state=42)	1.0
KNeighborsClassifier(n_neighbors=5)	1.0

3 Task C : Additional Features of YOLO v2 over YOLO v1

5 additional features present in YOLO v2 over YOLO v1 are:

1. **Usage of Darknet 19 Architecture:** YOLO v2 uses Darknet 19 architecture with 19 convolutional layers and 5 max pooling layers and a softmax layer for classification objects over YOLO v1's usage of 24 convolutional layers followed by 2 fully connected layers. Darknet is a neural network framework written in Clanguage and CUDA and is really fast in the task of object detection which is very important for predicting in real-time.

2. **Higher Resolution Classifier:** The input size in YOLO v2 was increased from 224x224 to 448x448. This increase in the input size of the image improved the MAP (mean average precision) upto 4% and was applied while training the YOLO v2 architecture DarkNet 19 on ImageNet dataset.

3. **Batch Normalization:** Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

   This normalises the input layer by altering slightly and scaling the activations appropriately and decreases the shift in unit value in the hidden layer and by doing so it improves the stability of the

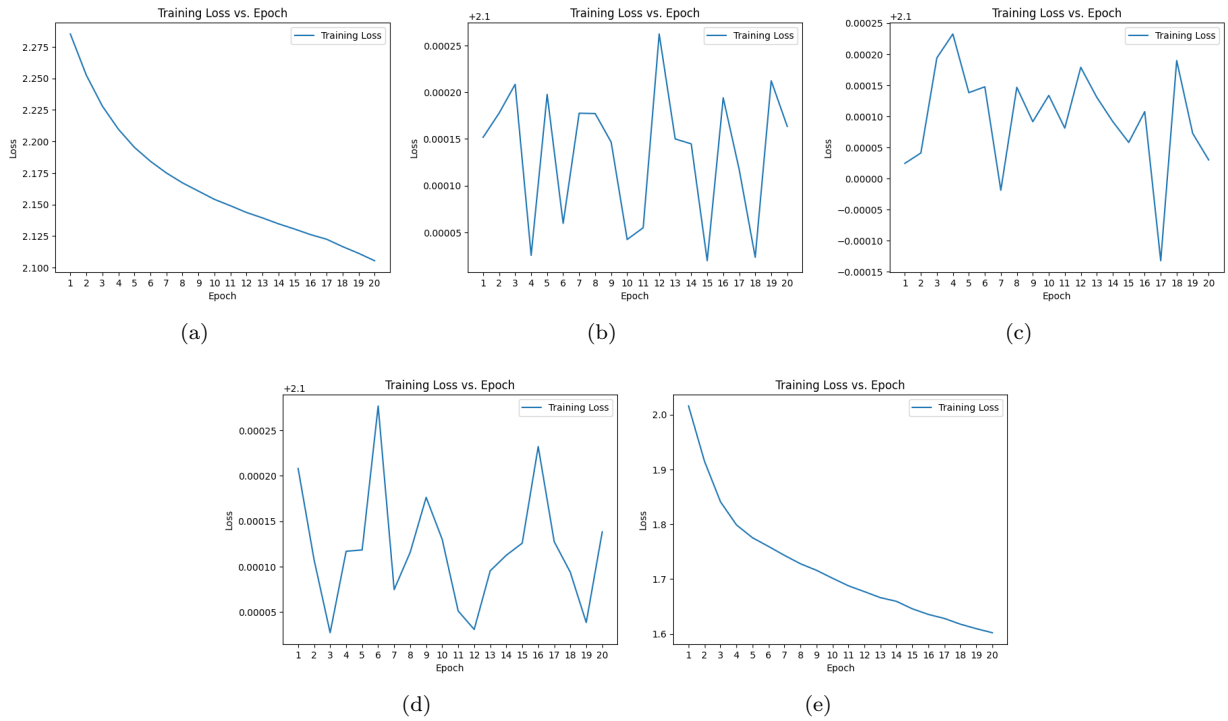

Figure 3: Snapshots showing Training Loss over Epochs for First Architecture with Sigmoid as activation function (a) Without Momentum (b) Momentum = 0.5 (c) Momentum = 0.7 (d) Momentum = 0.9 (e) With Adam

neural network. By adding batch normalization to convolutional layers in the architecture, the MAP (mean average precision) has been improved by 2%. It also helped the model regularise and overfitting has been reduced overall.

4. **Fine-Grained Features:** One of the main issues that had to be addressed in the YOLO v1 model was the detection of smaller objects present in the image. This was resolved in the YOLO v2 model which divides the image into 13x13 grid cells, smaller when compared to its previous version. This enables the YOLO v2 model to identify or localize smaller objects in the image and also be effective with larger objects.

5. **Anchor Boxes:** YOLOv2 introduced the concept of anchor boxes to improve the accuracy of bounding box predictions. Anchor boxes are pre-defined shapes of different aspect ratios and sizes. Instead of directly predicting the coordinates of bounding boxes, YOLOv2 predicts offsets from anchor boxes. Using anchor boxes allowed the model to better handle objects of different shapes and aspect ratios.

4 Task D : Car Counter using Object Tracking(SORT+DeepSORT)

The task requires you to collect a video of a traffic junction to implement a car counter to track and count the number of cars correctly.

4.1 Collection of Dataset and Initialization of models

The collected videos were read as images by reading the video frame by frame. The trackers were initialized with different parameters such as max age, nms overlap etc. These aided in accurately tracking and counting the number of cars for the video with reasonable accuracy. Ultralytics YoloV5l model and FasterRCNN from pytorch models were used for object detection.

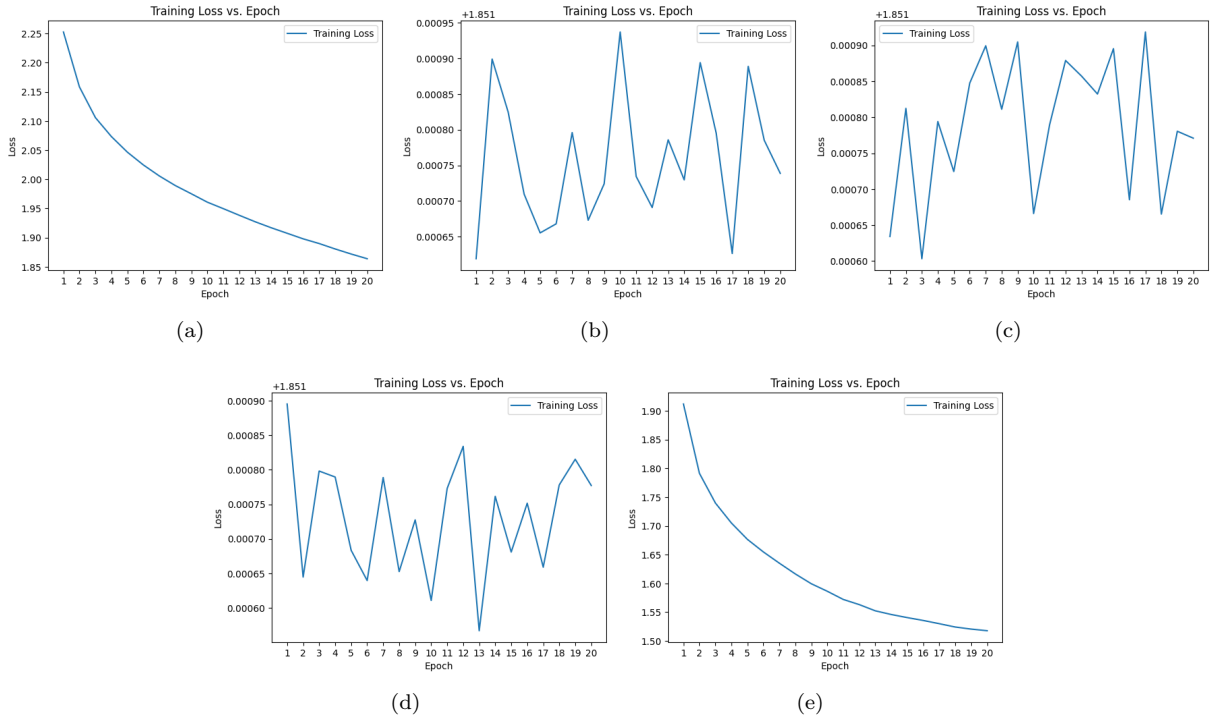

Figure 4: Snapshots showing Training Loss over Epochs for Second Architecture with ReLU as activation function (a) Without Momentum (b) Momentum = 0.5 (c) Momentum = 0.7 (d) Momentum = 0.9 (e) With Adam

4.2 Implementation Details

The car counter was implemented through the following steps

1. The object detection models and the multi object trackers are first initialized.
2. With the help of OpenCV the video is read as frames.
3. The frames are sent to one of the two object detection models. The results of the models are stored.
4. The bounding boxes predicted by the models is sent to the MOT trackers if the prediction score is more than a certain value and if the object detected is a car. Depending on the tracker the confidence score is either sent or not sent (Sort does not use confidence score while Deepsort does) .
5. The bounding boxes proposed by the trackers are then drawn on the frame while simultaneously keeping a track of the number of unique track ids until this point.
6. The count of unique track ids is also mentioned in the frame and is written onto the output file. Once this is done on all the frames the output video is generated.

4.3 Sort and Deepsort

Both these methods track objects using bounding boxes and tracker ids. While the use cases for both are the same there are some distinct differences which are noteworthy. Some of them are:

- Sort uses Kalman filter to track the movement of objects. The filter estimates the positon of the object by extrapolating it's motion and then the kalman filter updates its belief.
  Deepsort on the other hand, uses Kalman filter where the IOU overlap calculation is changed to

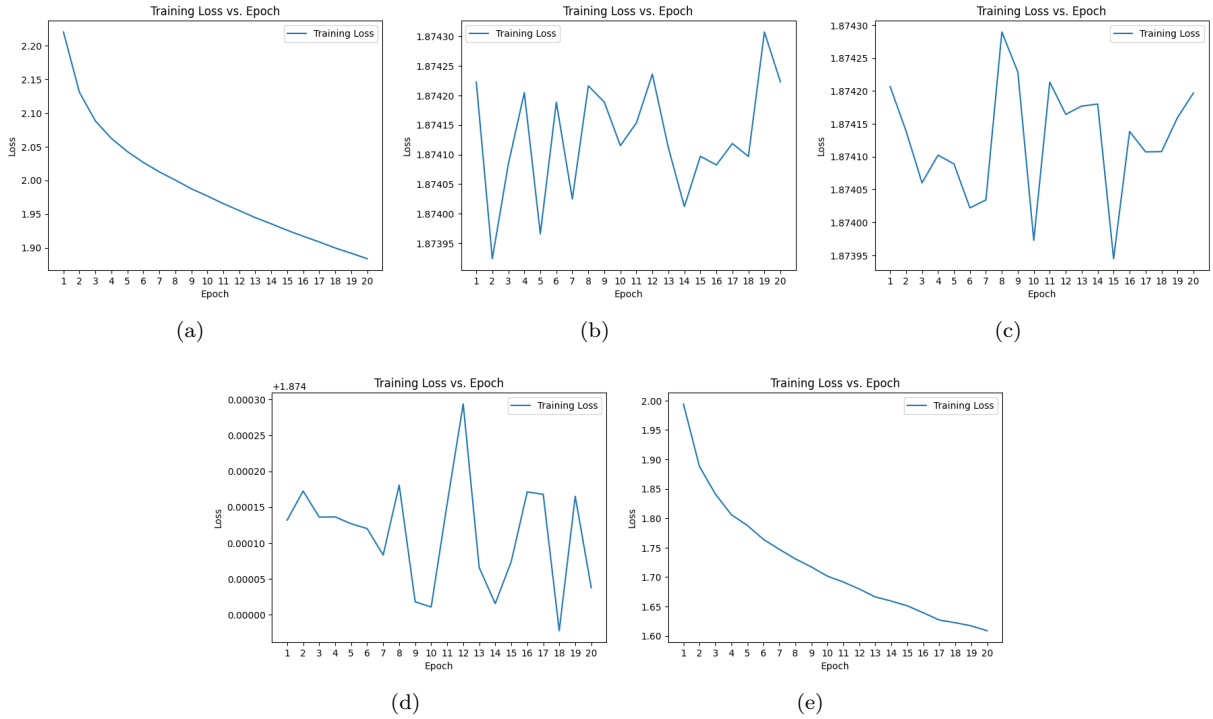

Figure 5: Snapshots showing Training Loss over Epochs for Second Architecture with Tanh as activation function (a) Without Momentum (b) Momentum = 0.5 (c) Momentum = 0.7 (d) Momentum = 0.9 (e) With Adam

Mahalanobis distance. It considers the uncertainty in prediction and measures distance even if there is no overlap.

- Deepsort tracks the cars better even with occlusion compared to SORT due to the usage of CNN to give a lower level embedding of the image. This CNN is the reason due to which Deepsort is able to track object better in crowded situations.
- Sort creates a new track identity when a new detection has an IOU below a specific threshold and is not added to an existing track while tracks are removed when no detection is assigned for a certain number of frames. Deepsort assigns a new detection for each track that scores below a certain age/frame threshold. The detection with the least cost is associated with the track. For all unmatched tracks, the algorithm runs an IOU matching like sort.

4.4 YoloV5 and Faster RCNN

Both the models were effective in identifying the cars. YoloV5 makes its prediction with a lower confidence score but does so at a faster rate. This leads to different sized detections of the same object which makes it harder for the MOT trackers to track correctly. Faster RCNN detects object with a much higher confidence score but takes a longer time to complete its calculations. This longer time ensures that the bounding boxes of the detections are good and don't lead to different detection sizes of the same object.

4.5 Observations and Results

The detections were done on a video with approximately 21 cars present. Four resulting videos were generated. Various parameters were tried and only the best parameters were chosen for the final run. Various

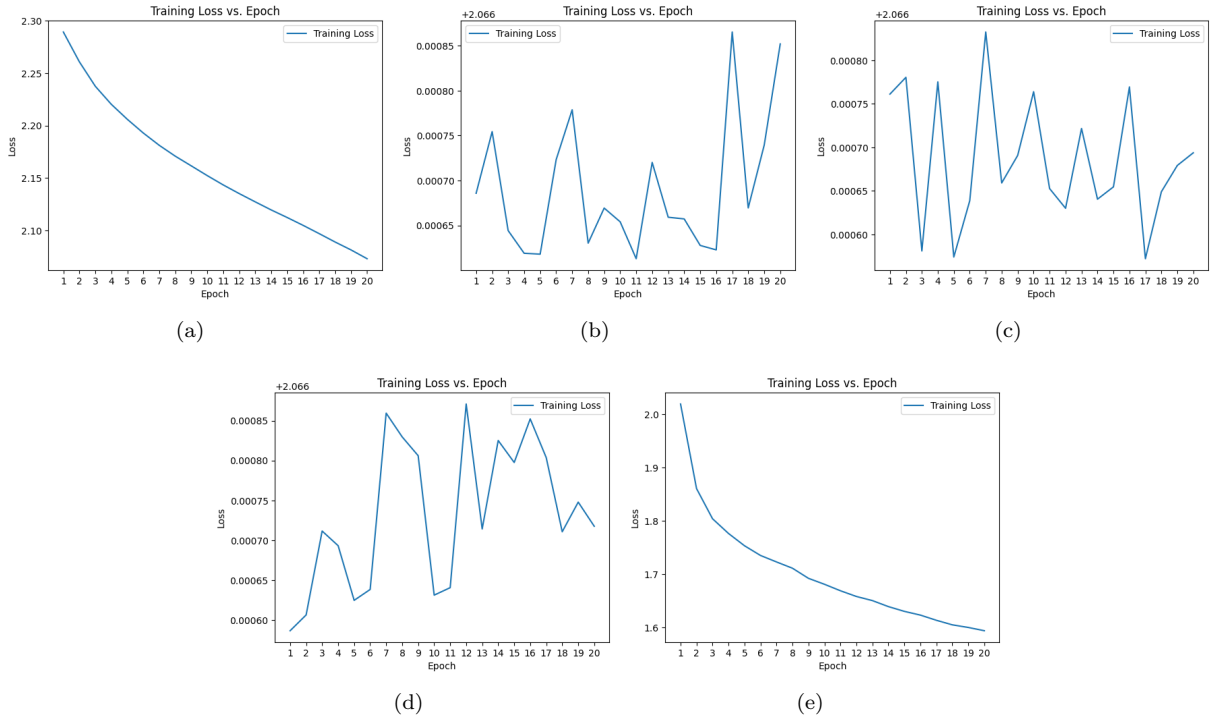

Figure 6: Snapshots showing Training Loss over Epochs for Second Architecture with Sigmoid as activation function (a) Without Momentum (b) Momentum = 0.5 (c) Momentum = 0.7 (d) Momentum = 0.9 (e) With Adam

confidence levels were tested to ensure maximum cars were detected while limiting the number of false predictions as well as multiple predictions of the same object. Max age was also tested to ensure that the cars were tracked only till a reasonable extent so as to delete the tracking bounding box. IOU threshold was also changed to ensure multiple trackers of the same objects are removed.

1. YoloV5 with deepsort :- For this we used a confidence threshold of 0.81 and initialized deepsort with a max age of 25 and a nms max overlap of 0.3. This gave a **car count of 25**
2. FasterRCNN with deepsort :- For this we used a confidence threshold of 0.993 (due to the high confidence scores) and initialized deepsort with max age of 20 and a max iou distance of 0.3. This gave a **car count of 24**.
3. YoloV5 with Sort :- For this we used a confidence threshold of 0.75 and initialized sort with iou threshold of 0.2 and max age of 600. This gave a **car count of 23**.
4. FasterRCNN with Sort :- For this we kept a confidence threshold of 0.97 and initialized sort with iou threshold of 0.2 and max age of 600. This gave a **car count of 21**.

Deepsort is useful when the object has to be accurately tracked with negligible error while Sort can be used when the tracking has to be on the fly owing to its light calculations. This has to be taken into consideration when deciding your criteria for the tracker and its specification.

4.6 Link to Tracking Videos

Car Tracking Videos