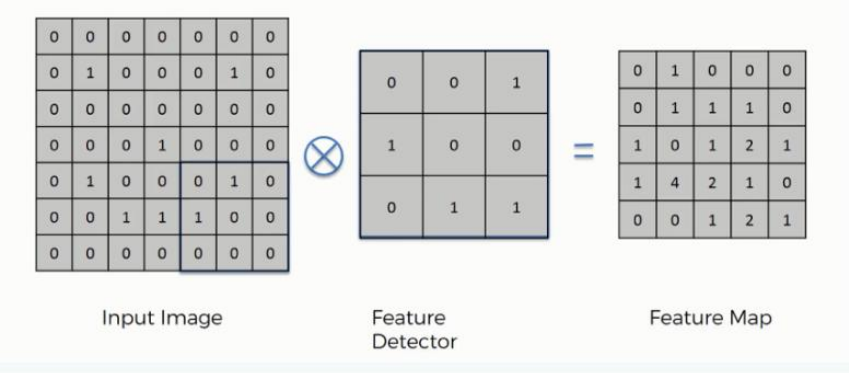
ECE 763 Report

Introduction:

Neural Networks that are based around convolutional layers are generally referred to as CNN's. They are generally followed by a dense layer and then a fully connected layer. The CNN is based around features extracted by the convolution layers.



Using different feature kernels, we extract different feature map. Every layer convolves the image and passes the feature to the next layer and so on. A fully connected layer means that all neurons are connected to each other.

The project uses a fairly newer model called Alexnet, which won the ImageNet challenge. It was one of the first deep convolutional networks to achieve an accuracy around 84.7%. The model consists of 5 convolutional layers and 3 fully connected layers along with RELU activation

Dataset:

1. **Dataset preprocessing**:

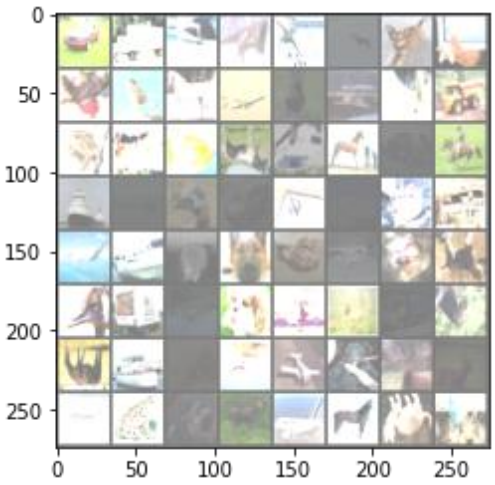
a. Data augmentation: We use data augmentation to reduce over fitting of the model. It includes few types of image transformation. The process includes image horizontal flip, vertical flip, brightness adjustment etc. This addresses the problems such as viewpoint, illumination etc.

b. Data normalization: Data normalization generally means dividing the image with the mean and standard deviation. This method would take care of the uneven illumination and makes the images more even.

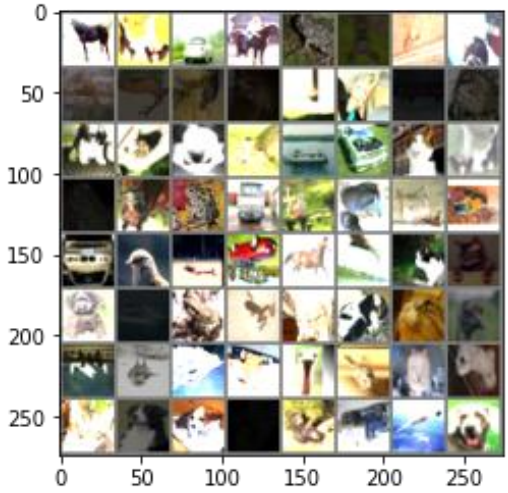
```
def load_dataset(self,dataset= None):
    # Using mean and std deviation
    normalization = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    # Using Augmentation and Normalization on the dataset
    self.train_transform = transforms.Compose([
                                transforms.RandomHorizontalFlip(),
                                transforms.RandomVerticalFlip(),
                                transforms.ColorJitter(brightness=2),
                                transforms.Resize((32,32)),
                                transforms.ToTensor(),
                                normalization,
                                        ])
```

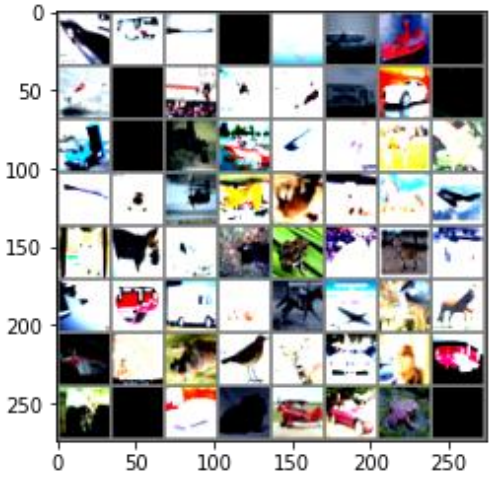
A. ***CIFAR10:***

i. **After Augmentation:**

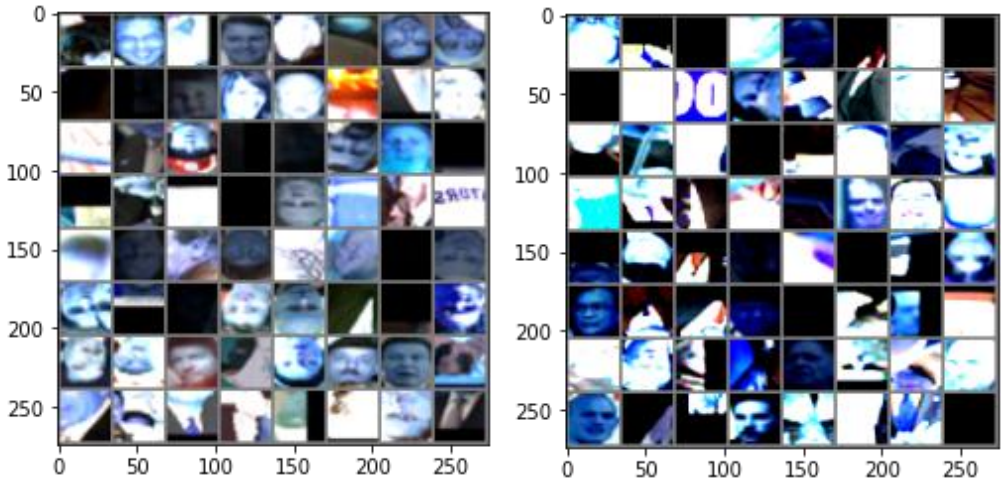


ii. **After Augmentation and Normalization:**





B. **FDDB Dataset**



2.Choosing NN Architecture:

Alexnet:

```
class AlexNet(nn.Module):
    def __init__(self, no_output):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, no_output),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

Babysitting the AlextNet for CIFAR10 dataset:

Model:AlextNet
Dataset: CIFAR10
Optimizer: SGD
Device: CUDA

1)**Ensuring the loss value is reasonable**:

```
100%|██████████| 1/1 [00:36<00:00, 36.04s/it]Epoch: 01 | Epoch Time: 0m 36s
        Training Loss: 2.302 | Validation Loss: 2.301 | Learning rate: 0.001| Reg:0
        Training Accuracy: 10.42% |  Validation Accuracy: 13.28%
```

For CIFAR-10 dataset, training labels = 10,
As we can see from the figure, the training loss is ~2.30 (i.e. loge(10))

2)**Considering a part of dataset to make the model overfit**:
10% of datasets is considered and trained to make the model overfit. Testing accuracy =1 is achieved.

3)Hyperparameter Optimization:

Coarse → fine train

In the first stage of the coarse train, we run only a few epochs = 5, with 20 different set of parameters to get a baseline idea of what params work.

```
max_count = 20

for count in tqdm(range(max_count)):
  LR = 10**random.uniform(-5,5)
  reg = 10**random.uniform(-3,-6)
  tr.train_model(trainset,validset,"cuda","SGD",5,LR,reg)
  print("-----------------------------------------------------------")
```

We optimize the log space over the highlighted values for LR and reg.

```
  0%|          | 0/5 [00:00<?, ?it/s]Device: cuda

 20%|          | 1/5 [00:36<02:24, 36.03s/it]Epoch: 01 | Epoch Time: 0m 36s
        Training Loss: nan | Validation Loss: nan | Learning rate: 510.314674028669
        Training Accuracy: 10.00% |  Validation Accuracy: 10.03%
Device: cuda

 40%|          | 2/5 [01:12<01:48, 36.07s/it]Epoch: 02 | Epoch Time: 0m 36s
        Training Loss: nan | Validation Loss: nan | Learning rate: 1000.4711538628923
        Training Accuracy: 10.00% |  Validation Accuracy: 10.03%
Device: cuda

Device: cuda

 40%|          | 2/5 [01:11<01:47, 35.71s/it]Epoch: 02 | Epoch Time: 0m 35s
        Training Loss: nan | Validation Loss: nan | Learning rate: 208.73233448444347
        Training Accuracy: 9.99% |  Validation Accuracy: 10.03%
Device: cuda
```

Here, the LR is way too high and thus the corresponding validation accuracy is low.

```
  0%|          | 0/5 [00:00<?, ?it/s]Device: cuda

 20%|          | 1/5 [00:36<02:24, 36.23s/it]Epoch: 01 | Epoch Time: 0m 36s
        Training Loss: 2.303 | Validation Loss: 2.303 | Learning rate: 6.331944386333944e-05
        Training Accuracy: 10.04% |  Validation Accuracy: 9.04%
Device: cuda
```

Even with a lower LR, the accuracy is less, since it's not optimal.

```
  0%|          | 0/5 [00:00<?, ?it/s]Device: cuda

 20%|          | 1/5 [00:36<02:24, 36.12s/it]Epoch: 01 | Epoch Time: 0m 36s
        Training Loss: 2.303 | Validation Loss: 2.302 | Learning rate: 3.0121400777449763e-05
        Training Accuracy: 9.63% |  Validation Accuracy: 11.25%
Device: cuda
```

As we keep on reducing the LR, the accuracy increases, but decreases after a while.

```
Device: cuda

100%|          | 5/5 [03:01<00:00, 36.28s/it]Epoch: 05 | Epoch Time: 0m 36s
        Training Loss: 1.425 | Validation Loss: 1.219 | Learning rate: 0.01241821833876381
        Training Accuracy: 48.53% |  Validation Accuracy: 55.02%
```

We narrowed the range to not be too low, nor too high but at 0.01.

Fine Train:

Here, since we narrowed the range, we run 10 times (parameters) with a number of epochs to be equal to 5 to find the optimal LR and Regularization value.

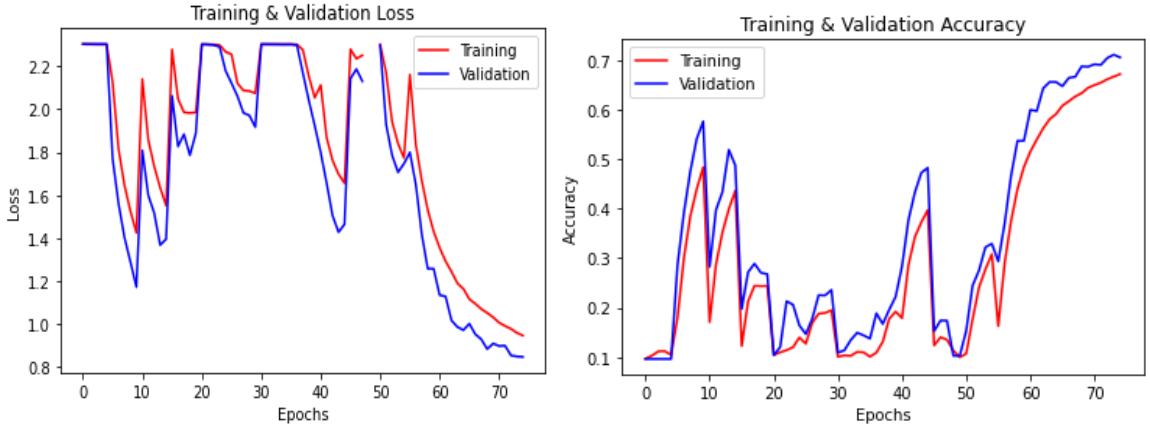
```
Device: cuda

100%|          | 5/5 [02:58<00:00, 35.62s/it]Epoch: 05 | Epoch Time: 0m 35s
        Training Loss: 1.424 | Validation Loss: 1.172 | Learning rate: 0.011332243268395086| Reg:0.00010304718875445554
        Training Accuracy: 48.41% |  Validation Accuracy: 57.68%
```

4) Final Train with the optimal parameters:

```
Device: cuda
100%|          | 20/20 [12:03<00:00, 36.16s/it]Epoch: 20 | Epoch Time: 0m 35s
        Training Loss: 0.946 | Validation Loss: 0.846 | Learning rate: 0.01| Reg:0.00011
        Training Accuracy: 67.25% |  Validation Accuracy: 70.64%
```

Validation accuracy with a LR=0.01 and Reg=0.00011 is 70.64 %

5)Training and validation curves:



6)Test loss and accuracy:

```
[52] test_loss, test_acc = tr.evaluate(tr.model, testset, criterion, device)
     print(f'Test Accuracy: {test_acc} | Test Loss: {test_loss}')

     Test Accuracy: 0.7069688498402555 | Test Loss: 0.859009473563764
```

The test accuracy is 70.7 % for CIFAR10 classification by AlexNet.

Babysitting the AlextNet for FDDB dataset:

Model:AlextNet
Dataset: FDDB
Optimizer: SGD
Device: cpu

1)Checking the first loss:

For FDDB dataset, training labels = 2,
As we can see from the figure, the training loss is ~0.69 (i.e. loge(2))

```
100%|██████████| 1/1 [00:03<00:00,  3.68s/it]Epoch: 01 | Epoch Time: 0m 3s
    Training Loss: 0.688 | Validation Loss: 0.659 | Learning rate: 0.0001
    Validation Accuracy: 57.78% |  Validation Accuracy: 65.00%
```

2)**Overfitting**:

```
  0%|          | 0/40 [00:00<?, ?it/s]Device: cpu
  2%|▏         | 1/40 [00:05<03:43,  5.72s/it]Epoch: 01 | Epoch Time: 0m 5s
    Training Loss: 0.690 | Validation Loss: 0.686 | Learning rate: 0.001
    Training Accuracy: 64.84% |  Validation Accuracy: 100.00%
Device: cpu
  5%|▌         | 2/40 [00:09<02:52,  4.53s/it]Epoch: 02 | Epoch Time: 0m 3s
    Training Loss: 0.679 | Validation Loss: 0.669 | Learning rate: 0.001
    Training Accuracy: 99.22% |  Validation Accuracy: 100.00%
Device: cpu
  8%|▊         | 3/40 [00:12<02:29,  4.04s/it]Epoch: 03 | Epoch Time: 0m 3s
    Training Loss: 0.660 | Validation Loss: 0.647 | Learning rate: 0.001
    Training Accuracy: 100.00% |  Validation Accuracy: 100.00%
Device: cpu
 10%|▉         | 4/40 [00:17<02:33,  4.26s/it]Epoch: 04 | Epoch Time: 0m 4s
    Training Loss: 0.637 | Validation Loss: 0.623 | Learning rate: 0.001
    Training Accuracy: 100.00% |  Validation Accuracy: 100.00%
Device: cpu
 12%|█▎        | 5/40 [00:23<02:53,  4.95s/it]Epoch: 05 | Epoch Time: 0m 6s
    Training Loss: 0.611 | Validation Loss: 0.596 | Learning rate: 0.001
    Training Accuracy: 100.00% |  Validation Accuracy: 100.00%
```

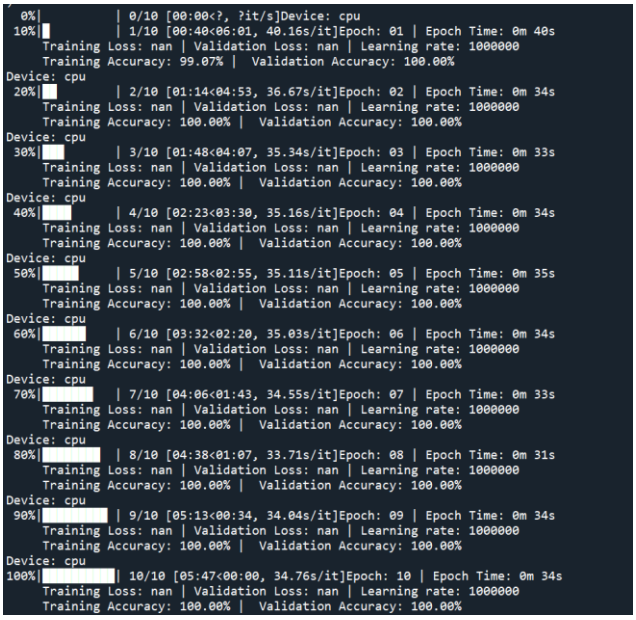
It can be observed that Training accuracy reaches 100%.

3) **With very low value of learning rate, loss value doesn't change**:

```
  0%|          | 0/10 [00:00<?, ?it/s]Device: cpu
 10%|▉         | 1/10 [00:33<05:05, 34.00s/it]Epoch: 01 | Epoch Time: 0m 33s
    Training Loss: 0.693 | Validation Loss: 0.694 | Learning rate: 1e-06
    Training Accuracy: 54.82% |  Validation Accuracy: 18.30%
Device: cpu
 20%|█▊        | 2/10 [01:08<04:34, 34.31s/it]Epoch: 02 | Epoch Time: 0m 34s
    Training Loss: 0.692 | Validation Loss: 0.693 | Learning rate: 1e-06
    Training Accuracy: 59.10% |  Validation Accuracy: 32.14%
Device: cpu
 30%|██▋       | 3/10 [01:43<04:02, 34.68s/it]Epoch: 03 | Epoch Time: 0m 35s
    Training Loss: 0.692 | Validation Loss: 0.693 | Learning rate: 1e-06
    Training Accuracy: 62.06% |  Validation Accuracy: 58.48%
Device: cpu
 40%|███▌      | 4/10 [02:20<03:33, 35.51s/it]Epoch: 04 | Epoch Time: 0m 36s
    Training Loss: 0.691 | Validation Loss: 0.692 | Learning rate: 1e-06
    Training Accuracy: 70.67% |  Validation Accuracy: 88.84%
Device: cpu
 50%|████▌     | 5/10 [03:04<03:12, 38.60s/it]Epoch: 05 | Epoch Time: 0m 44s
    Training Loss: 0.691 | Validation Loss: 0.692 | Learning rate: 1e-06
    Training Accuracy: 73.96% |  Validation Accuracy: 98.66%
Device: cpu
 60%|█████▍    | 6/10 [03:45<02:38, 39.55s/it]Epoch: 06 | Epoch Time: 0m 41s
    Training Loss: 0.690 | Validation Loss: 0.691 | Learning rate: 1e-06
    Training Accuracy: 78.18% |  Validation Accuracy: 100.00%
Device: cpu
 70%|██████▎   | 7/10 [04:24<01:57, 39.15s/it]Epoch: 07 | Epoch Time: 0m 38s
    Training Loss: 0.690 | Validation Loss: 0.691 | Learning rate: 1e-06
    Training Accuracy: 83.39% |  Validation Accuracy: 100.00%
Device: cpu
 80%|███████▏  | 8/10 [05:05<01:19, 39.73s/it]Epoch: 08 | Epoch Time: 0m 40s
    Training Loss: 0.689 | Validation Loss: 0.690 | Learning rate: 1e-06
    Training Accuracy: 86.51% |  Validation Accuracy: 100.00%
Device: cpu
 90%|████████▏ | 9/10 [05:49<00:41, 41.02s/it]Epoch: 09 | Epoch Time: 0m 43s
    Training Loss: 0.689 | Validation Loss: 0.690 | Learning rate: 1e-06
    Training Accuracy: 89.14% |  Validation Accuracy: 100.00%
Device: cpu
100%|█████████| 10/10 [06:27<00:00, 38.71s/it]Epoch: 10 | Epoch Time: 0m 38s
    Training Loss: 0.688 | Validation Loss: 0.690 | Learning rate: 1e-06
    Training Accuracy: 91.12% |  Validation Accuracy: 100.00%
```

It can be observed that there is no change in loss value as epochs changes.

4) **With high value of learning rate, loss explodes**:



It can be observed that loss values are reaching ‘nan’ and ‘inf’.

5)**Coarse Training:**

Coarse search is run with 5 epochs for 20 times by randomly choosing learning rate and regularization value:

```
max_count = 20

for count in tqdm(range(max_count)):
  LR = 10**random.uniform(-5,5)
  reg = 10**random.uniform(-3,-6)
  tr.train_model(trainset, validset, args.device, args.optimizer, 10,LR, reg)
```

Values obtained:

```
100%| | 10/10 [05:15<00:00, 31.54s/it]Epoch: 10 | Epoch Time: 0m 30s
     Training Loss: nan | Validation Loss: nan | Learning rate: 506.96587925190477
     Training Accuracy: 50.16% |  Validation Accuracy: 50.00%
```

```
100%| | 10/10 [05:55<00:00, 35.54s/it]Epoch: 10 | Epoch Time: 0m 40s
     Training Loss: 0.387 | Validation Loss: 0.270 | Learning rate: 0.018639206913321302
     Training Accuracy: 83.17% |  Validation Accuracy: 88.39%
```

```
100%|          | 10/10 [06:12<00:00, 37.22s/it]Epoch: 10 | Epoch Time: 0m 41s
     Training Loss: 0.675 | Validation Loss: 0.685 | Learning rate: 0.001179978436624557
     Training Accuracy: 58.99% |  Validation Accuracy: 59.82%
```

```
100%|          | 10/10 [05:50<00:00, 35.07s/it]Epoch: 10 | Epoch Time: 0m 57s
     Training Loss: 0.693 | Validation Loss: 0.693 | Learning rate: 0.00013629914167914256
     Training Accuracy: 49.84% |  Validation Accuracy: 50.00%
```

```
100%|          | 10/10 [05:17<00:00, 31.75s/it]Epoch: 10 | Epoch Time: 0m 31s
     Training Loss: nan | Validation Loss: nan | Learning rate: 69.33591359719358
     Training Accuracy: 49.51% |  Validation Accuracy: 50.00%
```

After analyzing the loss and accuracy values in coarse search, the Learning rate range is narrowed to (-4, -1) in power of 10 and regularization value is narrowed to range (-3,-5) power of 10.

6) Fine Training:

After running fine train, the optimal results are obtained at learning rate = 0.023 and regularization = 0.001

```
          | 10/10 [06:03<00:00, 36.37s/it]Epoch: 10 | Epoch Time: 0m 37s
Training Loss: 0.371 | Validation Loss: 0.291 | Learning rate: 0.023880999516035426
Training Accuracy: 83.94% |  Validation Accuracy: 86.61%
```

7) Final Training with optimal values:

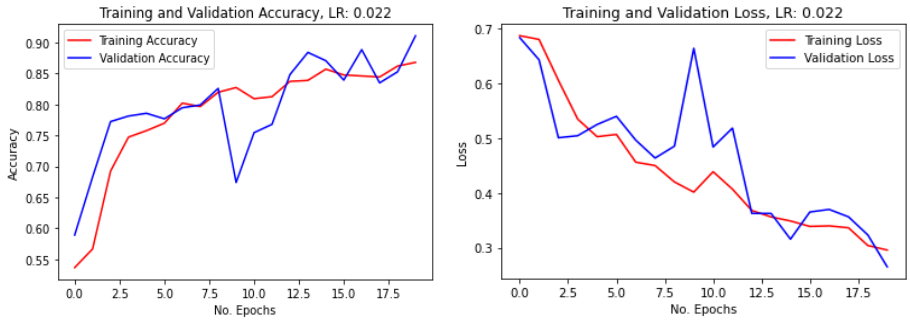
```
Device: cpu
100%|          | 20/20 [12:35<00:00, 37.76s/it]Epoch: 20 | Epoch Time: 0m 32s
     Training Loss: 0.223 | Validation Loss: 0.195 | Learning rate: 0.022
     Training Accuracy: 90.68% |  Validation Accuracy: 93.30%
```

8) Test accuracy and loss:

```
Test Accuracy: 0.9107142857142857 | Test Loss: 0.19411131526742662
```

Test accuracy is 91.07% and Test loss: 19.4%

Training and Validation accuracy and loss plots:



Training AlexNet on FDDB dataset by setting the optimizer to Adam:

Model:AlextNet
Dataset: FDDB
Optimizer: Adam
Device: cpu

Coarse training results:
After performing the coarse train, the Learning rate is narrowed down to range(-4,-1) power of 10 and regularization is narrowed to range(-3,-5) power of 10.

Fine training results:
Fine training gives best result when learning rate is around 0.001 and regularization is set to 0.001.

```
100%|          | 10/10 [04:32<00:00, 27.28s/it]Epoch: 10 | Epoch Time: 0m 30s
      Training Loss: 0.305 | Validation Loss: 0.269 | Learning rate: 0.0012833346514923682
      Training Accuracy: 88.43% |  Validation Accuracy: 90.18%
plot
```

Final Train with LR = 10^-3, reg = 10^-3:

FDDB final run after fine tuning we get highest accuracy:

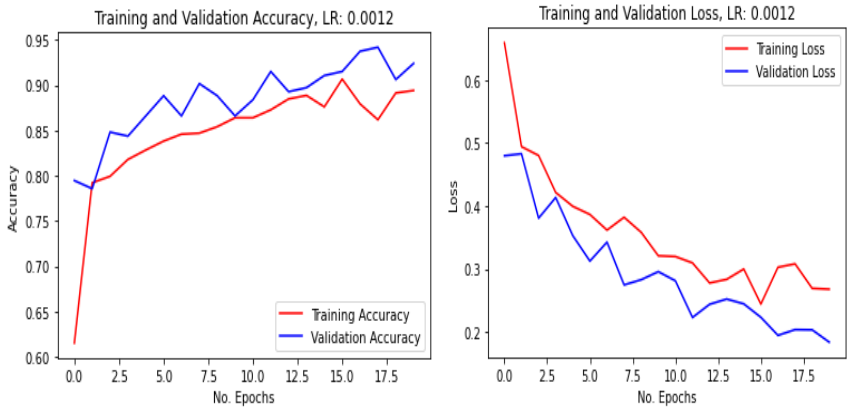
```
100%|          | 20/20 [09:58<00:00, 29.95s/it]Epoch: 20 | Epoch Time: 0m 29s
     Training Loss: 0.268 | Validation Loss: 0.184 | Learning rate: 0.0012
     Training Accuracy: 89.42% |  Validation Accuracy: 92.41%
```

Training accuracy = 89.42% and Training loss = 26.8%

Test accuracy and Test loss:

```
Test Accuracy: 0.9464285714285714 | Test Loss: 0.14549728270087922
```

Train and Validation curves:



Observations:

Dataset	Optimizer	Device	Test - Accuracy
CIFAR-10	SGD	cuda	70.7%
FDDB	SGD	cpu	91.07%
FDDB	Adam	cpu	94%

Conclusion:

An AlexNet based classifier is implemented and trained following babysitting method to classify FDDB dataset and CIFAR-10 dataset. It can be observed that the **AlexNet classifier** achieved **70.7% test accuracy** in classification of CIFAR-10 labels using **SGD Optimizer** on **cuda** device. The learning time is comparatively faster in cuda than cpu. The AlexNet classifier achieved **94% test accuracy** in FDDB classification with **Adam optimizer** which is better than **91.07% test accuracy** with **SGD optimizer**.

Team Contribution:

Aaron Mathew (asmathew): Pre-processing input data, report, modular code, Model implementation.
Snehapriyaa Mathiyalaghan(smathiy): Model training implementation, report, utility functions implementation.
Kiran Ganesh(kganesh4): Integration and Testing, report.