Luleå University of Technology

D7047E – Advanced Deep Learning

28 March 2023

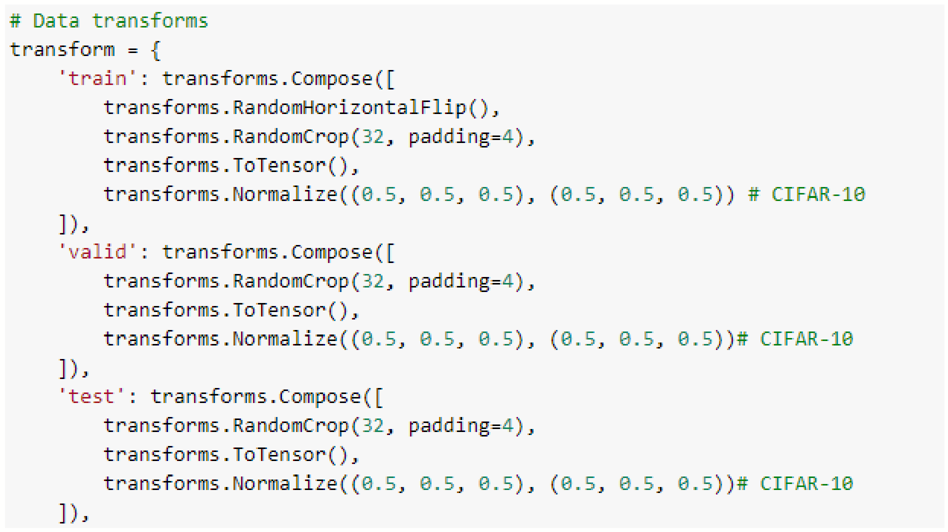
Exercise 1 | Group 1 | Berezin Ilya, Meenakshi Subhash Chippa

Contents

ConvNet model for image classification

Downloading and preparing CIFAR-10 dataset.

*Code was developed in Google Colab IDE due to availability of GPU's computational resources, please see the results in shared *.ipynb notebooks with generated outputs, including training epochs statistics and metrics plots.



```
# Data transforms
transform = {
    'train': transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # CIFAR-10
    ]),
    'valid': transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))# CIFAR-10
    ]),
    'test': transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))# CIFAR-10
    ]),
```

Figure.1.1 Image transform parameters.

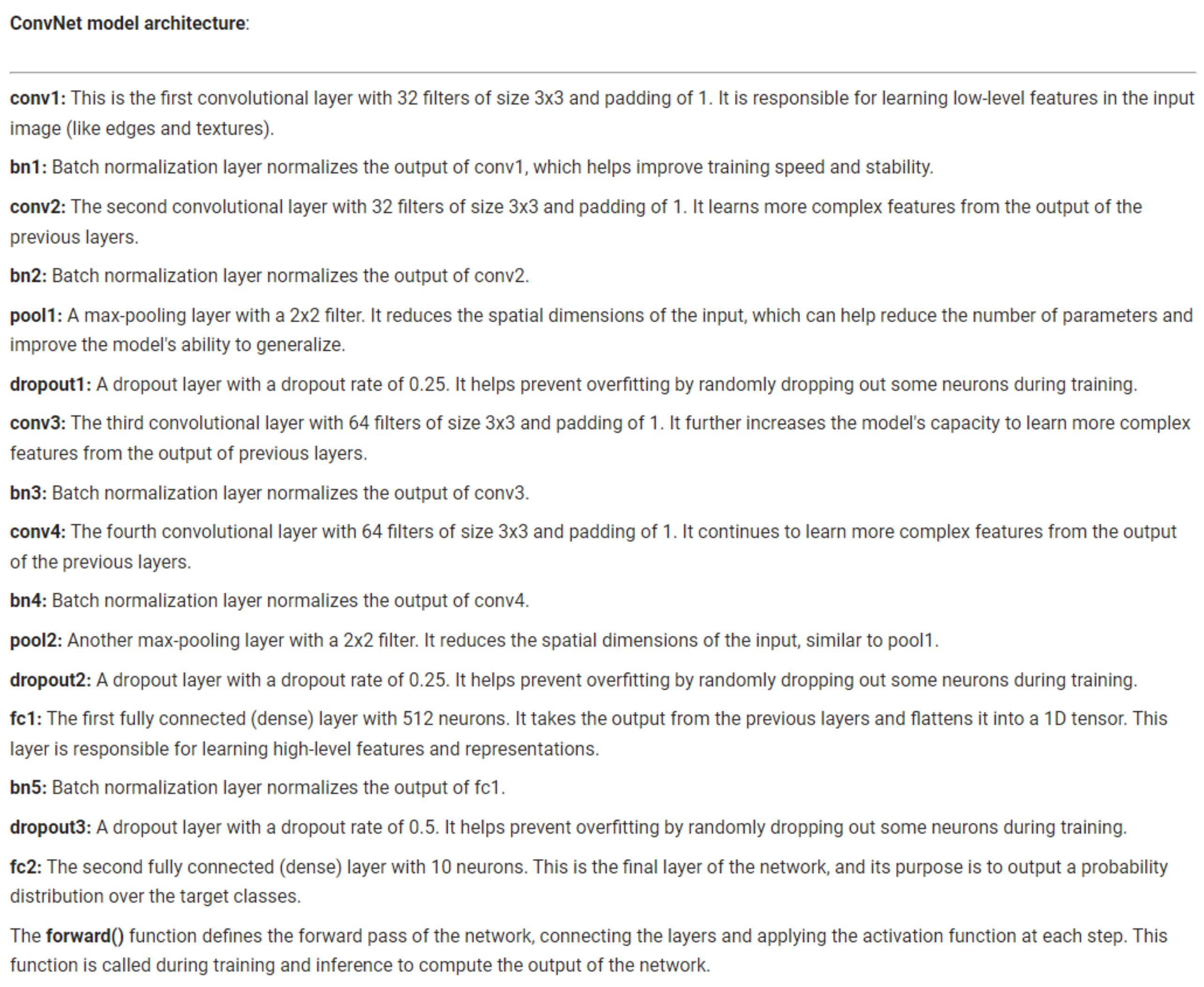
Performed a validation-test split of the data and use a manual seed so that we can reproduce the same split for the experiments. np.random.seed(42) was used for reproducibility. Test dataset was split as 20% for the test and 80% for the validation dataset.

Obtained splits verified on being disjoint.

Batch size is set to 128 to speed-up the computation using available RAM resources.

Model (ConvNet) has been trained displaying progress, current accuracies, and losses for both training and validation dataset for different hyperparameters (ADAM, SGD) optimizers, (LeakyReLU, Tanh) activation functions and 0.0001 learning rate.

ConvNet model training and evaluation



ConvNet model architecture:

conv1: This is the first convolutional layer with 32 filters of size 3x3 and padding of 1. It is responsible for learning low-level features in the input image (like edges and textures).

bn1: Batch normalization layer normalizes the output of conv1, which helps improve training speed and stability.

conv2: The second convolutional layer with 32 filters of size 3x3 and padding of 1. It learns more complex features from the output of the previous layers.

bn2: Batch normalization layer normalizes the output of conv2.

pool1: A max-pooling layer with a 2x2 filter. It reduces the spatial dimensions of the input, which can help reduce the number of parameters and improve the model's ability to generalize.

dropout1: A dropout layer with a dropout rate of 0.25. It helps prevent overfitting by randomly dropping out some neurons during training.

conv3: The third convolutional layer with 64 filters of size 3x3 and padding of 1. It further increases the model's capacity to learn more complex features from the output of previous layers.

bn3: Batch normalization layer normalizes the output of conv3.

conv4: The fourth convolutional layer with 64 filters of size 3x3 and padding of 1. It continues to learn more complex features from the output of the previous layers.

bn4: Batch normalization layer normalizes the output of conv4.

pool2: Another max-pooling layer with a 2x2 filter. It reduces the spatial dimensions of the input, similar to pool1.

dropout2: A dropout layer with a dropout rate of 0.25. It helps prevent overfitting by randomly dropping out some neurons during training.

fc1: The first fully connected (dense) layer with 512 neurons. It takes the output from the previous layers and flattens it into a 1D tensor. This layer is responsible for learning high-level features and representations.

bn5: Batch normalization layer normalizes the output of fc1.

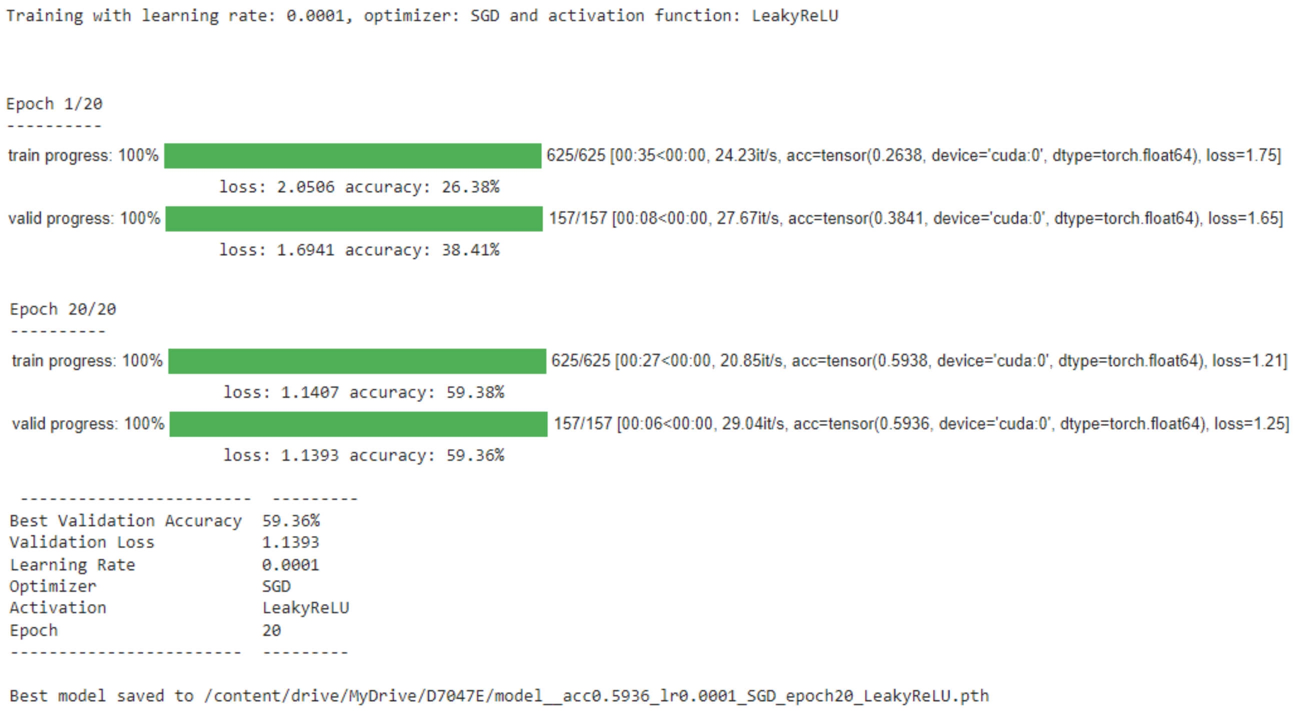
dropout3: A dropout layer with a dropout rate of 0.5. It helps prevent overfitting by randomly dropping out some neurons during training.

fc2: The second fully connected (dense) layer with 10 neurons. This is the final layer of the network, and its purpose is to output a probability distribution over the target classes.

The **forward()** function defines the forward pass of the network, connecting the layers and applying the activation function at each step. This function is called during training and inference to compute the output of the network.

Figure.1.2 ConvNet model layers

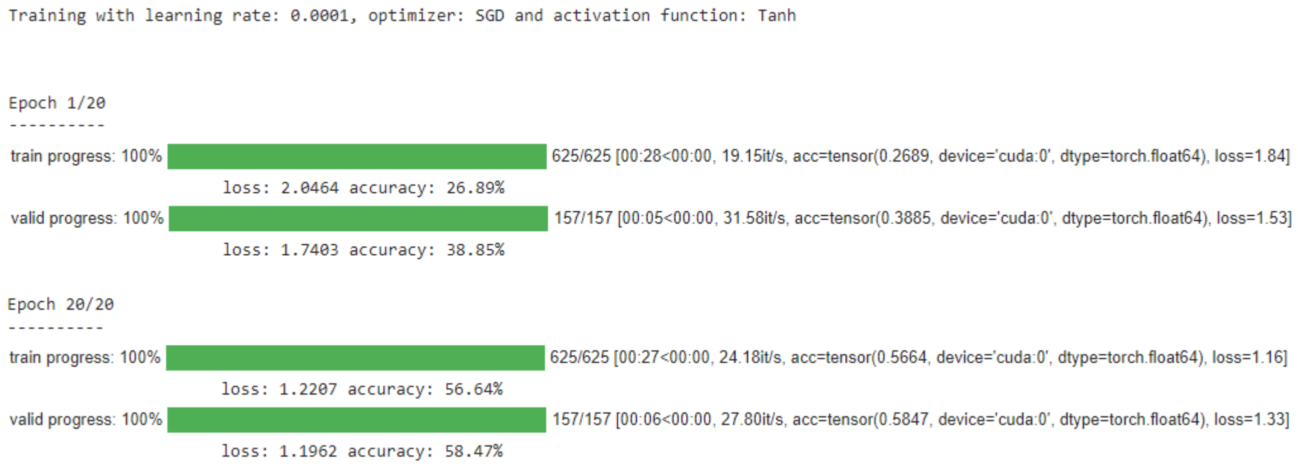
ConvNet for image classification / 0.0001 learning rate, SGD optimizer and LeakyReLU activation.



```
Training with learning rate: 0.0001, optimizer: SGD and activation function: LeakyReLU


Epoch 1/20
----------
train progress: 100% 625/625 [00:35<00:00, 24.23it/s, acc=tensor(0.2638, device='cuda:0', dtype=torch.float64), loss=1.75]
                  loss: 2.0506 accuracy: 26.38%
valid progress: 100% 157/157 [00:08<00:00, 27.67it/s, acc=tensor(0.3841, device='cuda:0', dtype=torch.float64), loss=1.65]
                  loss: 1.6941 accuracy: 38.41%

Epoch 20/20
----------
train progress: 100% 625/625 [00:27<00:00, 20.85it/s, acc=tensor(0.5938, device='cuda:0', dtype=torch.float64), loss=1.21]
                  loss: 1.1407 accuracy: 59.38%
valid progress: 100% 157/157 [00:06<00:00, 29.04it/s, acc=tensor(0.5936, device='cuda:0', dtype=torch.float64), loss=1.25]
                  loss: 1.1393 accuracy: 59.36%

 ------------------------  ---------
Best Validation Accuracy  59.36%
Validation Loss           1.1393
Learning Rate             0.0001
Optimizer                 SGD
Activation                LeakyReLU
Epoch                     20
------------------------  ---------

Best model saved to /content/drive/MyDrive/D7047E/model__acc0.5936_lr0.0001_SGD_epoch20_LeakyReLU.pth
```

Figure.1.3 ConvNet model parameters on CIFAR-10 using SGD optimizer and LeakyReLU activation.

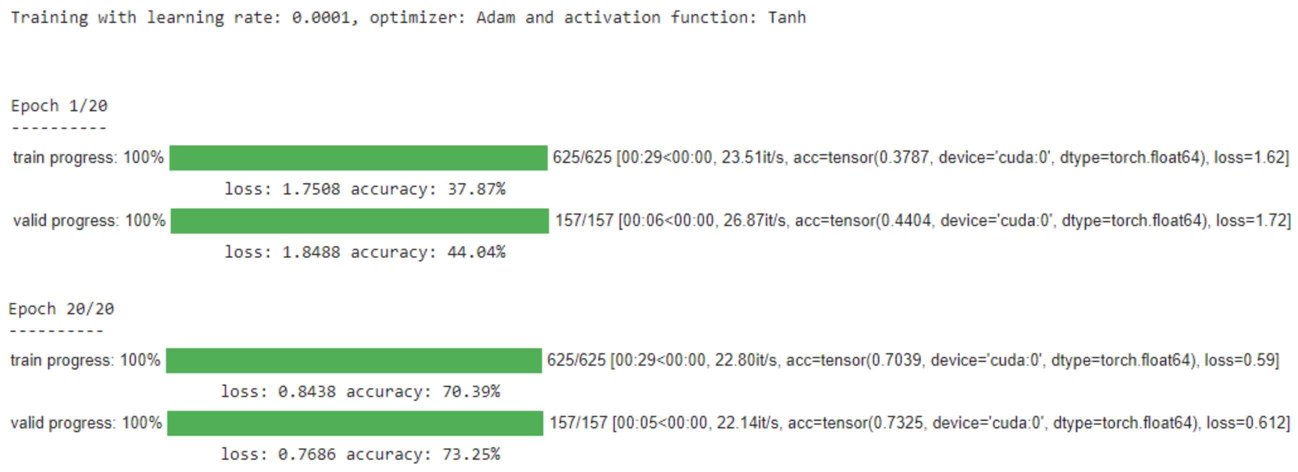
ConvNet for image classification / 0.0001 learning rate, SGD optimizer and Tanh activation.



```
Training with learning rate: 0.0001, optimizer: SGD and activation function: Tanh


Epoch 1/20
----------
train progress: 100% 625/625 [00:28<00:00, 19.15it/s, acc=tensor(0.2689, device='cuda:0', dtype=torch.float64), loss=1.84]
                  loss: 2.0464 accuracy: 26.89%
valid progress: 100% 157/157 [00:05<00:00, 31.58it/s, acc=tensor(0.3885, device='cuda:0', dtype=torch.float64), loss=1.53]
                  loss: 1.7403 accuracy: 38.85%

Epoch 20/20
----------
train progress: 100% 625/625 [00:27<00:00, 24.18it/s, acc=tensor(0.5664, device='cuda:0', dtype=torch.float64), loss=1.16]
                  loss: 1.2207 accuracy: 56.64%
valid progress: 100% 157/157 [00:06<00:00, 27.80it/s, acc=tensor(0.5847, device='cuda:0', dtype=torch.float64), loss=1.33]
                  loss: 1.1962 accuracy: 58.47%
```

Figure.1.4 ConvNet model parameters on CIFAR-10 using SGD optimizer and Tanh activation.

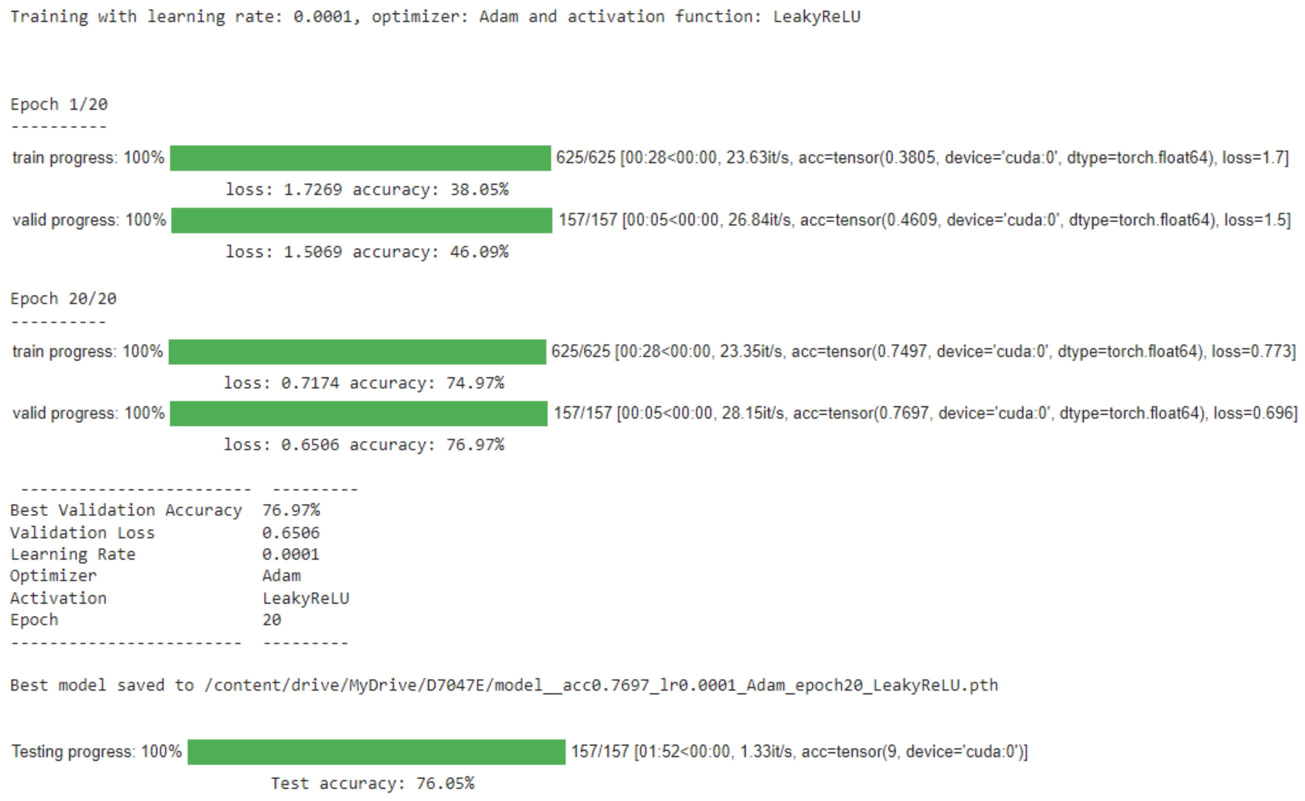
ConvNet for image classification / 0.0001 learning rate, Adam optimizer and Tanh activation.



```
Training with learning rate: 0.0001, optimizer: Adam and activation function: Tanh


Epoch 1/20
----------
train progress: 100%  625/625 [00:29<00:00, 23.51it/s, acc=tensor(0.3787, device='cuda:0', dtype=torch.float64), loss=1.62]
          loss: 1.7508 accuracy: 37.87%
valid progress: 100%  157/157 [00:06<00:00, 26.87it/s, acc=tensor(0.4404, device='cuda:0', dtype=torch.float64), loss=1.72]
          loss: 1.8488 accuracy: 44.04%

Epoch 20/20
----------
train progress: 100%  625/625 [00:29<00:00, 22.80it/s, acc=tensor(0.7039, device='cuda:0', dtype=torch.float64), loss=0.59]
          loss: 0.8438 accuracy: 70.39%
valid progress: 100%  157/157 [00:05<00:00, 22.14it/s, acc=tensor(0.7325, device='cuda:0', dtype=torch.float64), loss=0.612]
          loss: 0.7686 accuracy: 73.25%
```

Figure.1.5 ConvNet model parameters on CIFAR-10 using Adam optimizer and Tanh activation.

ConvNet for image classification / 0.0001 learning rate, Adam optimizer and LeakyReLU activation.



```
Training with learning rate: 0.0001, optimizer: Adam and activation function: LeakyReLU


Epoch 1/20
----------
train progress: 100%  625/625 [00:28<00:00, 23.63it/s, acc=tensor(0.3805, device='cuda:0', dtype=torch.float64), loss=1.7]
          loss: 1.7269 accuracy: 38.05%
valid progress: 100%  157/157 [00:05<00:00, 26.84it/s, acc=tensor(0.4609, device='cuda:0', dtype=torch.float64), loss=1.5]
          loss: 1.5069 accuracy: 46.09%

Epoch 20/20
----------
train progress: 100%  625/625 [00:28<00:00, 23.35it/s, acc=tensor(0.7497, device='cuda:0', dtype=torch.float64), loss=0.773]
          loss: 0.7174 accuracy: 74.97%
valid progress: 100%  157/157 [00:05<00:00, 28.15it/s, acc=tensor(0.7697, device='cuda:0', dtype=torch.float64), loss=0.696]
          loss: 0.6506 accuracy: 76.97%

 -----------------------  ---------
Best Validation Accuracy  76.97%
Validation Loss           0.6506
Learning Rate             0.0001
Optimizer                 Adam
Activation                LeakyReLU
Epoch                     20
-----------------------   ---------

Best model saved to /content/drive/MyDrive/D7047E/model__acc0.7697_lr0.0001_Adam_epoch20_LeakyReLU.pth


Testing progress: 100%  157/157 [01:52<00:00, 1.33it/s, acc=tensor(9, device='cuda:0')]
          Test accuracy: 76.05%
```

Figure.1.6 ConvNet model parameters on CIFAR-10 using Adam optimizer and LeakyReLU activation.

ConvNet model metrics plotted after training on CIFAR-10.

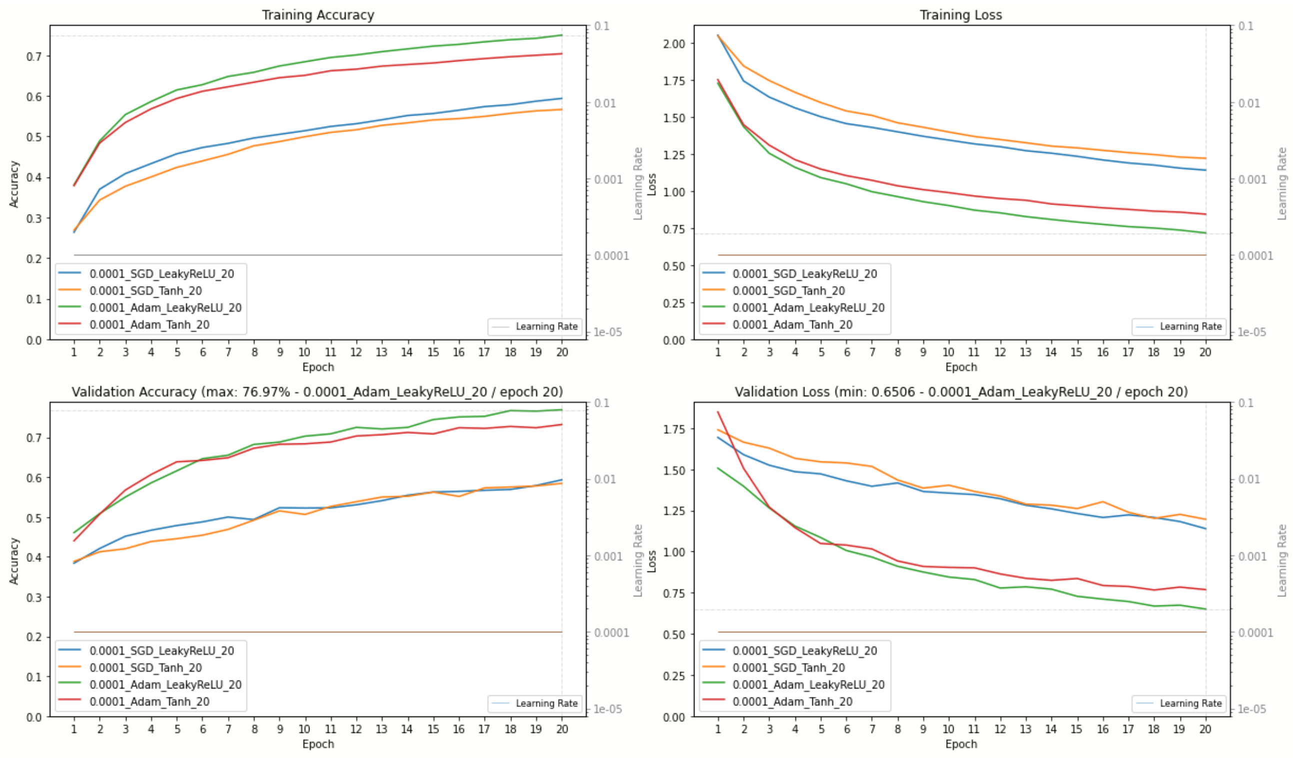
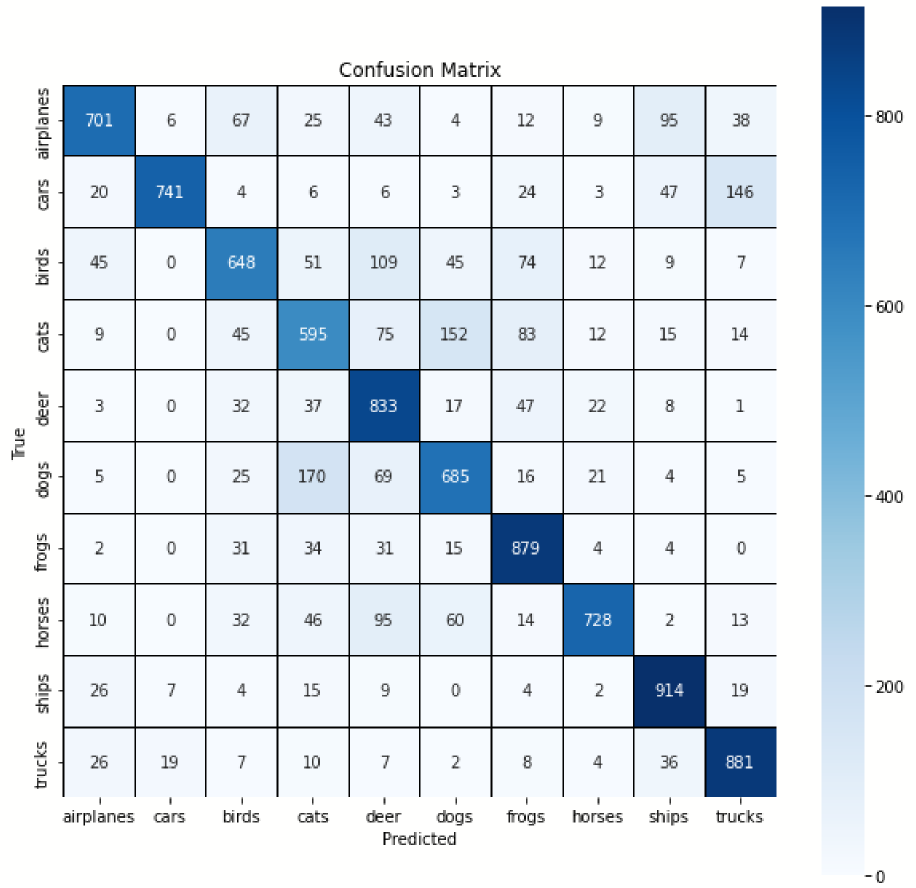


Figure.1.7 ConvNet model metrics plotted after training with different hyperparameters.

ConvNet model confusion matrix and classification report.



True \ Predicted	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	701	6	67	25	43	4	12	9	95	38
cars	20	741	4	6	6	3	24	3	47	146
birds	45	0	648	51	109	45	74	12	9	7
cats	9	0	45	595	75	152	83	12	15	14
deer	3	0	32	37	833	17	47	22	8	1
dogs	5	0	25	170	69	685	16	21	4	5
frogs	2	0	31	34	31	15	879	4	4	0
horses	10	0	32	46	95	60	14	728	2	13
ships	26	7	4	15	9	0	4	2	914	19
trucks	26	19	7	10	7	2	8	4	36	881

```
Classification Report:

              precision    recall  f1-score   support

   airplanes       0.83      0.70      0.76      1000
        cars       0.96      0.74      0.84      1000
       birds       0.72      0.65      0.68      1000
        cats       0.60      0.59      0.60      1000
        deer       0.65      0.83      0.73      1000
        dogs       0.70      0.69      0.69      1000
       frogs       0.76      0.88      0.81      1000
      horses       0.89      0.73      0.80      1000
       ships       0.81      0.91      0.86      1000
      trucks       0.78      0.88      0.83      1000

    accuracy                           0.76     10000
   macro avg       0.77      0.76      0.76     10000
weighted avg       0.77      0.76      0.76     10000
```

Figure.1.7 ConvNet model confusion matrix and classification report for the best model.