Image Classification and Visualizations with Convolutional Neural Networks – ImageNet10

Name	**Shilpa Gopalakrishna**

QUESTION I

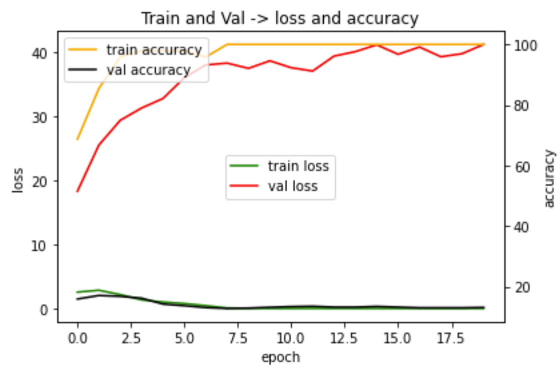
1.1 Single-batch training

1.1.1. Display graph 1.1.1 (training & validation loss over training epochs) and briefly explain what is happening and why?

Explanation:

- Below is the chart of a simple linear model's Train/Validation loss & accuracy

Graph 1.1.1



- Linear model with no hidden layers, the input image tensor (128*128*3) is passed to Linear layer and classes are predicted as output
- Batch size=48
- Trained the model on 1 batch of train data

Brief on the task:

- Here Linear model with no hidden layers is used
- The training of model is done over single train batch of 48 records

- **What is happening?**

- The model is overfitting. As we can see from the chart that the Train loss reduces very fast and is almost flat, but the validation loss is too high & also the validation accuracy is too low. Hence, we can say that the model is not able to generalize and is over fitting (train accuracy too high, train loss almost 0 but the validation loss too high & validation accuracy too low)

- **Why is the model overfitting?**
  - For a problem like image classification, linear model is too simple and is not able to recognize patterns in the image because of the linear function. Also, single batch of train set for training is not enough for the model to learn the various image patterns.
  - The model training can be stopped at 14th or 15th epoch as the training loss is flat

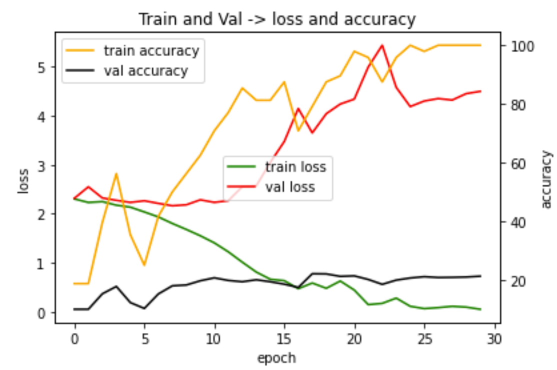
```
Epoch: 0 |  T_Loss: 2.8423 | T_Acc: 68.7500 | V_Loss: 18.3225 | V_Acc: 15.9356
Epoch: 1 |  T_Loss: 2.1669 | T_Acc: 85.4167 | V_Loss: 25.5245 | V_Acc: 17.1016
Epoch: 2 |  T_Loss: 1.3379 | T_Acc: 95.8333 | V_Loss: 29.3966 | V_Acc: 16.8240
Epoch: 3 |  T_Loss: 1.0000 | T_Acc: 97.9167 | V_Loss: 31.3202 | V_Acc: 16.2132
Epoch: 4 |  T_Loss: 0.7635 | T_Acc: 97.9167 | V_Loss: 32.8066 | V_Acc: 14.2699
Epoch: 5 |  T_Loss: 0.4280 | T_Acc: 97.9167 | V_Loss: 36.0638 | V_Acc: 13.7146
Epoch: 6 |  T_Loss: 0.0682 | T_Acc: 95.8333 | V_Loss: 38.0536 | V_Acc: 13.1038
Epoch: 7 |  T_Loss: 0.0216 | T_Acc: 100.0000 | V_Loss: 38.3361 | V_Acc: 12.8262
Epoch: 8 |  T_Loss: 0.0008 | T_Acc: 100.0000 | V_Loss: 37.5193 | V_Acc: 12.9373
Epoch: 9 |  T_Loss: 0.0021 | T_Acc: 100.0000 | V_Loss: 38.6971 | V_Acc: 13.2149
Epoch: 10 |  T_Loss: 0.0059 | T_Acc: 100.0000 | V_Loss: 37.6200 | V_Acc: 13.4370
Epoch: 11 |  T_Loss: 0.0060 | T_Acc: 100.0000 | V_Loss: 37.0957 | V_Acc: 13.5480
Epoch: 12 |  T_Loss: 0.0031 | T_Acc: 100.0000 | V_Loss: 39.4311 | V_Acc: 13.2704
Epoch: 13 |  T_Loss: 0.0013 | T_Acc: 100.0000 | V_Loss: 40.1329 | V_Acc: 13.2704
Epoch: 14 |  T_Loss: 0.0005 | T_Acc: 100.0000 | V_Loss: 41.2002 | V_Acc: 13.4925
Epoch: 15 |  T_Loss: 0.0003 | T_Acc: 100.0000 | V_Loss: 39.7169 | V_Acc: 13.2704
Epoch: 16 |  T_Loss: 0.0002 | T_Acc: 100.0000 | V_Loss: 40.8549 | V_Acc: 13.0483
Epoch: 17 |  T_Loss: 0.0001 | T_Acc: 100.0000 | V_Loss: 39.3309 | V_Acc: 13.0483
Epoch: 18 |  T_Loss: 0.0002 | T_Acc: 100.0000 | V_Loss: 39.8164 | V_Acc: 13.0483
Epoch: 19 |  T_Loss: 0.0005 | T_Acc: 100.0000 | V_Loss: 41.2632 | V_Acc: 13.1594
```

1.1.2 Display graph 1.1.2 (training & validation loss over training epochs, with modified architecture) and explain how and why it shows that the model is overfitting the training batch.

Explanation:

- Below is the chart of model with hidden layers Train/Validation loss & accuracy

Graph 1.1.2



Brief on the task:

- Here the model architecture contains 3 hidden layers and 2 fully connected layers

- The training of model is done over single train batch of 48 records

- **Why is the model overfitting?**
  - Single batch of train set for training is not enough for the model to learn the various image patterns as the model generally ends up overfitting and fails to generalize.
  - The model would have done a bit better even without optimization if we had trained the model with a more number of records because more the data used in training, the model adjusts the weights for more number & variety of records

- **How to say that the model is overfitting?**
  - The train loss is reducing, and train accuracy reached 100% which is good, but as we have trained on only one batch of train data, the validation accuracy is bad.
  - As the model is doing well on training set after evaluation & is doing bad on predictions on unseen records from validation set, we can say that the model is overfitting

```
Epoch: 0 |  T_Loss: 2.2291 | T_Acc: 18.7500 | V_Loss: 2.3138 | V_Acc: 9.9944
Epoch: 1 |  T_Loss: 2.2461 | T_Acc: 18.7500 | V_Loss: 2.5484 | V_Acc: 9.9944
Epoch: 2 |  T_Loss: 2.1735 | T_Acc: 39.5833 | V_Loss: 2.3209 | V_Acc: 15.2693
Epoch: 3 |  T_Loss: 2.1331 | T_Acc: 56.2500 | V_Loss: 2.2722 | V_Acc: 17.7679
Epoch: 4 |  T_Loss: 2.0353 | T_Acc: 35.4167 | V_Loss: 2.2296 | V_Acc: 12.2710
Epoch: 5 |  T_Loss: 1.9335 | T_Acc: 25.0000 | V_Loss: 2.2634 | V_Acc: 10.2721
Epoch: 6 |  T_Loss: 1.8000 | T_Acc: 41.6667 | V_Loss: 2.2101 | V_Acc: 15.2138
Epoch: 7 |  T_Loss: 1.6778 | T_Acc: 50.0000 | V_Loss: 2.1615 | V_Acc: 17.9900
Epoch: 8 |  T_Loss: 1.5489 | T_Acc: 56.2500 | V_Loss: 2.1813 | V_Acc: 18.2676
Epoch: 9 |  T_Loss: 1.4040 | T_Acc: 62.5000 | V_Loss: 2.2826 | V_Acc: 19.7113
Epoch: 10 |  T_Loss: 1.2209 | T_Acc: 70.8333 | V_Loss: 2.2305 | V_Acc: 20.7107
Epoch: 11 |  T_Loss: 1.0097 | T_Acc: 77.0833 | V_Loss: 2.2656 | V_Acc: 19.8223
Epoch: 12 |  T_Loss: 0.8079 | T_Acc: 85.4167 | V_Loss: 2.5495 | V_Acc: 19.3781
Epoch: 13 |  T_Loss: 0.6609 | T_Acc: 81.2500 | V_Loss: 2.5682 | V_Acc: 20.0444
Epoch: 14 |  T_Loss: 0.6320 | T_Acc: 81.2500 | V_Loss: 3.0460 | V_Acc: 19.3781
Epoch: 15 |  T_Loss: 0.4683 | T_Acc: 87.5000 | V_Loss: 3.4735 | V_Acc: 18.5453
Epoch: 16 |  T_Loss: 0.5843 | T_Acc: 70.8333 | V_Loss: 4.1470 | V_Acc: 17.4348
Epoch: 17 |  T_Loss: 0.4766 | T_Acc: 79.1667 | V_Loss: 3.6489 | V_Acc: 22.0988
Epoch: 18 |  T_Loss: 0.6254 | T_Acc: 87.5000 | V_Loss: 4.0402 | V_Acc: 22.0433
Epoch: 19 |  T_Loss: 0.4411 | T_Acc: 89.5833 | V_Loss: 4.2379 | V_Acc: 21.2104
Epoch: 20 |  T_Loss: 0.1433 | T_Acc: 97.9167 | V_Loss: 4.3378 | V_Acc: 21.3770
Epoch: 21 |  T_Loss: 0.1690 | T_Acc: 95.8333 | V_Loss: 4.9936 | V_Acc: 20.1555
Epoch: 22 |  T_Loss: 0.2762 | T_Acc: 87.5000 | V_Loss: 5.4383 | V_Acc: 18.4897
Epoch: 23 |  T_Loss: 0.1092 | T_Acc: 95.8333 | V_Loss: 4.5784 | V_Acc: 19.9334
Epoch: 24 |  T_Loss: 0.0616 | T_Acc: 100.0000 | V_Loss: 4.1861 | V_Acc: 20.6552
Epoch: 25 |  T_Loss: 0.0817 | T_Acc: 97.9167 | V_Loss: 4.2994 | V_Acc: 21.0994
Epoch: 26 |  T_Loss: 0.1090 | T_Acc: 100.0000 | V_Loss: 4.3500 | V_Acc: 20.8218
Epoch: 27 |  T_Loss: 0.0923 | T_Acc: 100.0000 | V_Loss: 4.3195 | V_Acc: 20.8773
Epoch: 28 |  T_Loss: 0.0485 | T_Acc: 100.0000 | V_Loss: 4.4505 | V_Acc: 20.9883
Epoch: 29 |  T_Loss: 0.0237 | T_Acc: 100.0000 | V_Loss: 4.4972 | V_Acc: 21.2660
```

1.1.3 Fill in table 1.1.3 (your adjusted architecture after single-batch training), adding rows and columns as necessary.

Explanation:

Adjusted network architecture without Optimization includes:

- 3 Conv2d Layers, each with a Relu activation & a maxpool layer
- Followed by 2 Fully connected Linear layer with Relu activation
- And the final Linear layer that converts 512 features to 10 class

The filter channels extracts the features from the image and maxpool & stride reduce the image dimension at the same time making sure that the filter extracts the features from the image. At the end of the linear layer, the feature size is reduced to 512 and extracting final class from those features.

Input channels	Output channels	Layer type	Kernel size	Pad	Stride
3	32	**Conv2d**	3	1	1
		Relu			
		maxpool	4		2
32	40	**Conv2d**	2	0	1
		Relu			
		maxpool	2		2
40	56	**Conv2d**	3	0	2
		Relu			
		maxpool	3		2
		Flatten	7*7*56		
		Linear	7*7*56, 1024		
		Relu			
		Linear	1024, 512		
		Relu			
		Linear	512, 10		

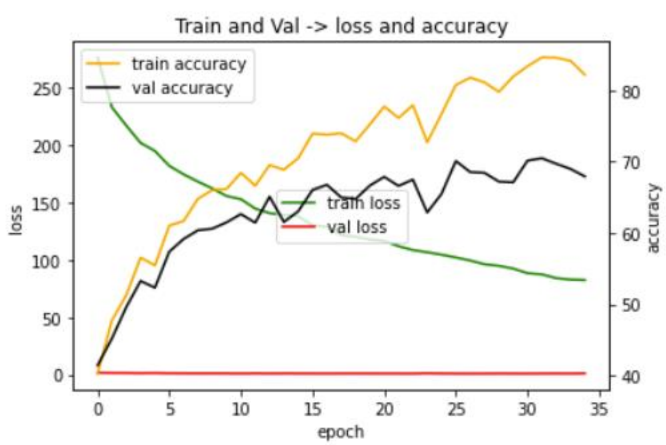
1.2 Fine-tuning on full dataset [18 marks]

1.2.1 Display graph 1.2.1 and indicate what the optimal number of training epochs is and why.

Explanation:

- Below is the chart of finetuned model's Train/Validation loss & accuracy

Graph 1.2.1



- After finetuning the model & on training the model with full train dataset for 35 epochs, we get below accuracy & loss on 35th epoch.

- Train accuracy=82.16%, Train loss=0.5
- Validation accuracy=69%, Validation loss= 1.1

- **Evaluating the model:** We can see from the below plot that the train & validation accuracy is gradually increasing. Train loss is reducing constantly whereas validation loss is changing in minute scale.

- **Optimal number of epochs:** By increasing the epochs and training with more epochs, could have improved the test & validation accuracy. Also, we could have trained the network initially with a higher learning rate and gradually reduce the learning rate so that we could have reached the global minima sooner. With the current network, it might take >20 more epochs to reach minimal loss as the rate of change in loss is in small scale

```
Epoch: 0 |  T_Loss: 1.6810 | T_Acc: 40.2639 | V_Loss: 1.6366 | V_Acc: 41.4214
Epoch: 1 |  T_Loss: 1.5107 | T_Acc: 47.7361 | V_Loss: 1.5124 | V_Acc: 45.1971
Epoch: 2 |  T_Loss: 1.3809 | T_Acc: 51.2639 | V_Loss: 1.4676 | V_Acc: 49.5836
Epoch: 3 |  T_Loss: 1.2580 | T_Acc: 56.5139 | V_Loss: 1.3040 | V_Acc: 53.2482
Epoch: 4 |  T_Loss: 1.2855 | T_Acc: 55.4306 | V_Loss: 1.4134 | V_Acc: 52.3043
Epoch: 5 |  T_Loss: 1.1128 | T_Acc: 61.0000 | V_Loss: 1.2154 | V_Acc: 57.3570
Epoch: 6 |  T_Loss: 1.1178 | T_Acc: 61.6528 | V_Loss: 1.1921 | V_Acc: 59.1338
Epoch: 7 |  T_Loss: 1.0203 | T_Acc: 64.7778 | V_Loss: 1.1353 | V_Acc: 60.3554
Epoch: 8 |  T_Loss: 0.9964 | T_Acc: 65.9861 | V_Loss: 1.1632 | V_Acc: 60.5775
Epoch: 9 |  T_Loss: 0.9772 | T_Acc: 66.1528 | V_Loss: 1.1269 | V_Acc: 61.4659
Epoch: 10 |  T_Loss: 0.9227 | T_Acc: 68.4167 | V_Loss: 1.0338 | V_Acc: 62.6319
Epoch: 11 |  T_Loss: 0.9603 | T_Acc: 66.5972 | V_Loss: 1.1717 | V_Acc: 61.4103
Epoch: 12 |  T_Loss: 0.8675 | T_Acc: 69.5000 | V_Loss: 1.0608 | V_Acc: 65.0750
Epoch: 13 |  T_Loss: 0.9403 | T_Acc: 68.8472 | V_Loss: 1.1249 | V_Acc: 61.5214
Epoch: 14 |  T_Loss: 0.8389 | T_Acc: 70.4583 | V_Loss: 1.0780 | V_Acc: 63.0205
Epoch: 15 |  T_Loss: 0.7601 | T_Acc: 73.9167 | V_Loss: 1.0406 | V_Acc: 66.0189
Epoch: 16 |  T_Loss: 0.7525 | T_Acc: 73.7639 | V_Loss: 1.0001 | V_Acc: 66.7407
Epoch: 17 |  T_Loss: 0.7451 | T_Acc: 73.9583 | V_Loss: 1.0772 | V_Acc: 64.9639
Epoch: 18 |  T_Loss: 0.7966 | T_Acc: 72.8194 | V_Loss: 1.0831 | V_Acc: 64.7418
Epoch: 19 |  T_Loss: 0.6996 | T_Acc: 75.2083 | V_Loss: 1.0528 | V_Acc: 66.6297
Epoch: 20 |  T_Loss: 0.6389 | T_Acc: 77.7222 | V_Loss: 1.0224 | V_Acc: 67.8512
Epoch: 21 |  T_Loss: 0.6816 | T_Acc: 76.1111 | V_Loss: 1.0158 | V_Acc: 66.5741
Epoch: 22 |  T_Loss: 0.6370 | T_Acc: 77.9028 | V_Loss: 0.9547 | V_Acc: 67.4625
Epoch: 23 |  T_Loss: 0.7804 | T_Acc: 72.7083 | V_Loss: 1.2269 | V_Acc: 62.8540
Epoch: 24 |  T_Loss: 0.6720 | T_Acc: 76.6528 | V_Loss: 1.0481 | V_Acc: 65.4636
Epoch: 25 |  T_Loss: 0.5488 | T_Acc: 80.7361 | V_Loss: 0.9755 | V_Acc: 70.0722
Epoch: 26 |  T_Loss: 0.5403 | T_Acc: 81.7639 | V_Loss: 0.9295 | V_Acc: 68.5175
Epoch: 27 |  T_Loss: 0.5556 | T_Acc: 81.0972 | V_Loss: 0.9843 | V_Acc: 68.4064
Epoch: 28 |  T_Loss: 0.5847 | T_Acc: 79.7639 | V_Loss: 1.0268 | V_Acc: 67.1849
Epoch: 29 |  T_Loss: 0.5208 | T_Acc: 81.9167 | V_Loss: 0.9965 | V_Acc: 67.0738
Epoch: 30 |  T_Loss: 0.4836 | T_Acc: 83.3611 | V_Loss: 0.9486 | V_Acc: 70.1277
Epoch: 31 |  T_Loss: 0.4551 | T_Acc: 84.5972 | V_Loss: 1.0133 | V_Acc: 70.4609
Epoch: 32 |  T_Loss: 0.4454 | T_Acc: 84.5556 | V_Loss: 1.0686 | V_Acc: 69.6835
Epoch: 33 |  T_Loss: 0.4627 | T_Acc: 84.1250 | V_Loss: 0.9814 | V_Acc: 68.9617
Epoch: 34 |  T_Loss: 0.5162 | T_Acc: 82.1667 | V_Loss: 1.1698 | V_Acc: 67.9067
```

1.2.2 Describe in detail your fine-tuning process on the complete dataset, including any adjustments you made to the network or training process to increase prediction accuracy. Explain why these adjustments increased accuracy.

Explanation:

- Model Architecture with fine-tuned network structure looks as below
    - 3 layers of Conv2D each with Relu & Maxpool layers
    - Dropout layer
    - Batch Normalization
    - Final fully connected layers

```
net_opt = nn.Sequential(
    nn.Conv2d(3,32, kernel_size=3,stride=1,padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=4, stride=2),

    nn.Conv2d(32,40, kernel_size=2, stride=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Dropout(0.05),

    nn.Conv2d(40,56, kernel_size=3,stride=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Dropout(0.1),

    nn.Flatten(),
    nn.Linear(7*7*56,1024),
    nn.BatchNorm1d(num_features=1024),
    nn.ReLU(),
    nn.Dropout(0.1),
    nn.Linear(1024,512),

    nn.ReLU(),
    nn.Linear(512,10)
```

- Various optimization steps to improver performance are as below:
    1. **Added multiple layers in the network** – Added additional Conv2D layers in the network. Optimized the network by initially starting with higher kernel size with padding and gradually reducing the kernel size. Having a deeper network allows the model to learn more complex features and patterns and with optimal tuning, will provide better predictions than a shallow network.
    2. **Augmentation** – Data augmentation was done on training data by randomly flipping the training data, adding jitter & random affine. But this did not increase the accuracy much on validation data as our validation set does not include any images that are drastically different from training set in terms of angles and orientation in the image. However, this helped the model training with more variety of data that were scaled, flipped, or rotated.

```python
data_transform = transforms.Compose([
        transforms.Resize(128),
        transforms.CenterCrop(128),
        transforms.ToTensor(),
        transforms.ColorJitter(hue=0.2, saturation=0.2, brightness=0.2),
        transforms.RandomAffine(degrees=10, translate=(0.1,0.1), scale=(0.9,1.1)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.Normalize(NORM_MEAN, NORM_STD),
    ])
```

3. **Added Dropout layers** -After fine tuning the model with additional hidden layers, dropout was introduced at 3 layers. Initially, trained the network with 2 dropout with a probability of 0.3 and 0.5, but the accuracy was too low. Hence, reduced the dropout probability to 0.1 in 2 places and introduced another dropout with 0.05 probability.
4. **Batch normalization**: Included batch normalization in the fully connected layer that normalizes the output of a layer before being fed to next layer

1.2.3 Display two confusion matrices 1.2.3 (one each for complete validation set and complete training set) for your final trained model and interpret what is shown.

Explanation:

- The train accuracy & loss on complete training data is 82.6% & 0.5 respectively
- The validation accuracy & loss on complete validation data is 68.18% & 1.02 respectively

```python
loss_fulltrain, acc_fulltrain, all_pred_train,all_actual_train = evaluate_loader(train_loader, netopt_fulltrain)
loss_val, acc_val,all_pred_val,all_actual_val = evaluate_loader(valid_loader, netopt_fulltrain)
print('T_Loss: %.4f | T_Acc: %.4f | V_Loss: %.4f | V_Acc: %.4f' %(loss_fulltrain,acc_fulltrain, loss_val , acc_val))
```

```
T_Loss: 0.5031 | T_Acc: 82.6111 | V_Loss: 1.0261 | V_Acc: 68.1843
```

Confusion matrix:

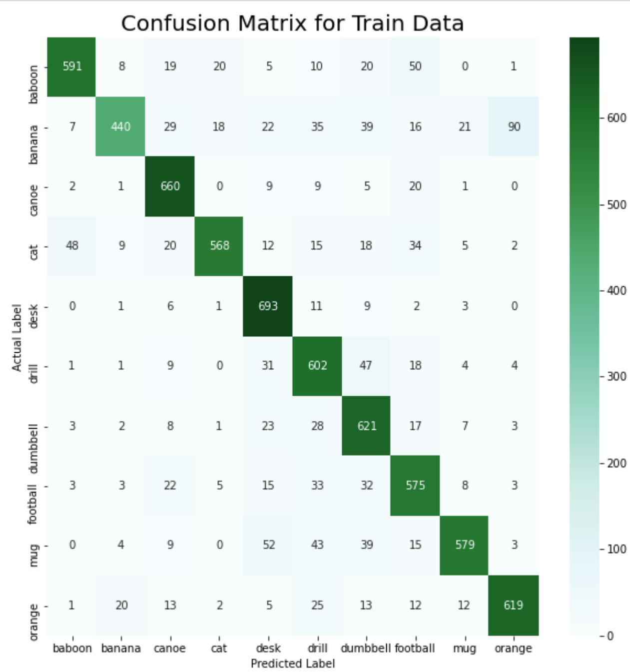
- Confusion matrix can be used to pictographically represent the performance of classification model.
- The confusion matrix compares the actual label/class/target with the predicted label/class/target
- In the below confusion matrix plot, there are 10 targets where column indicates the actual class & rows indicates the predicted class

Confusion matrix for Train dataset:

- Below plot represents confusion matrix after the trained model is evaluated on training dataset

- The diagonal elements represent the number of correct predictions for a particular class/target (represented by row/column header)
- An example to understand below matrix is:
  - ➢ The class “baboon” is correctly predicted as “baboon” by the model in 591 instances ie 1st row, 1st column
  - ➢ The class “baboon” is wrongly predicted to other class 65 times, ie rows from 2nd to 10th in 1st column
  - ➢ Other class were wrong classified as “baboon” in 132 instances ie First row columns from 2nd to 10th

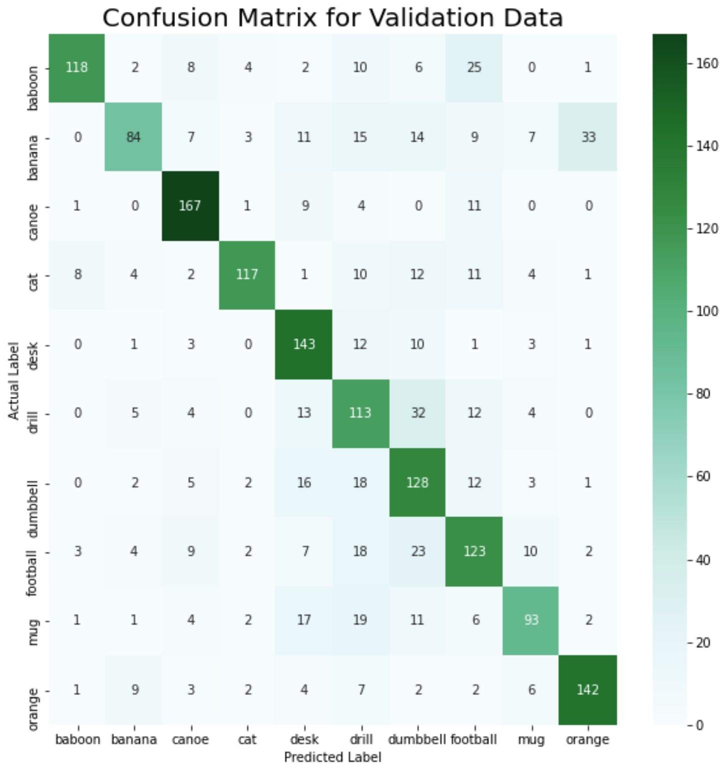
```
plot_confusion_matrix_train_data(all_pred_train,all_actual_train)
```



Confusion matrix for validation dataset:

- Below plot represents confusion matrix after the trained model is evaluated on validation dataset
- Like training set above, one example using validation test on understanding below matrix is:
  - ➢ The class “baboon” is correctly predicted as “baboon” by the model 118 times ie 1st row, 1st column
  - ➢ The class “baboon” is wrongly predicted to other class 14 times, ie rows from 2nd to 10th in 1st column
  - ➢ Other class were wrong classified as “baboon” in 77 instances ie First row columns from 2nd to 10th

```
plot_confusion_matrix_validation_data(all_pred_val,all_actual_val)
```



1.3 Evaluation and code

1.3.1 Please include *[my_student_username]_test_preds.csv* with your final submission. [8 marks]

Done

1.3.2 Please submit all relevant code you wrote for Question I in Python file *[my_student_username]_q1.py*. No need to include the config or ImageNet10 files. [13 marks]

Done

No response needed here.

QUESTION II

2.1 Preparing the pre-trained network

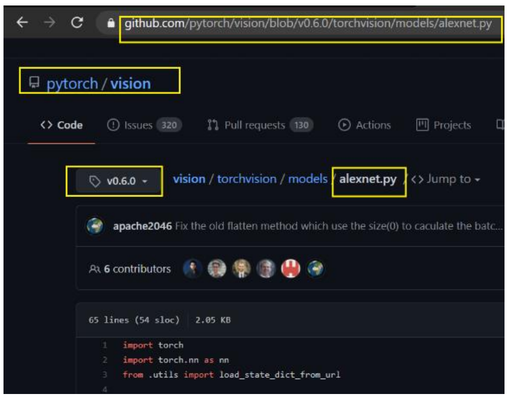
2.1.1 Read through the provided template code for the AlexNet model *alexnet.py*. What exactly is being loaded in line 59? [2 marks]

Explanation:

model = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet', pretrained=True)

In the above line of code, torch.hub.load imports/loads a pre-trained model with pretrained weights from specific github repository.

- 1st argument - name of the GIT repository ie pytorch/vision with a version tag of 0.6.0
- 2nd argument – model to be imported from the specified model repository ie “alexnet”
- 3rd argument – Load the pretrained weights from the repository



2.1.2 Write the code in *explore.py* after line 50 to read in the image specified in the variable `args.image_path` and pass it through a single forward pass of the pre-trained AlexNet model.
Code added

2.1.3 Fill in function `extract_filter()` after line 84 extracting the filters from a given layer of the pre-trained AlexNet.
Code added

2.1.4 Fill in function `extract_feature_maps()` after line 105 extracting the feature maps from the convolutional layers of the pre-trained AlexNet.
Code added

Please submit all your Question II code in a Python file *[my_student_username]_explore.py*.

No response needed here.

2.1.5 Describe in words, not code, how you ensure that your filters and feature maps are pairs; that the feature maps you extract correspond to the given filter.

Explanation:

filter: The filter index and its dimension are printed by looping the convD layers of the Alexnet model

Index 0 - torch.Size([64, 3, 11, 11]) (out_channel, in_channels, image dinsension of 11*11)

Index 3 - torch.Size([192, 64, 5, 5])
Index 6 - torch.Size([384, 192, 3, 3])
Index 8 - torch.Size([256, 384, 3, 3])
Index 10 - torch.Size([256, 256, 3, 3])

feature: The feature index at same index as ConvD and its dimension are printed by looping the image after passing through a layer at certain index of the Alexnet model
Index 0 - torch.Size([1, 64, 93, 124]) (number of images, out_channels after passing through a layer, image dimension of 93*124)
Index 3 - torch.Size([1, 192, 46, 61])
Index 6 - torch.Size([1, 384, 22, 30])
Index 8 - torch.Size([1, 256, 22, 30])
Index 10 -torch.Size([1, 256, 22, 30])

- The size of the featuremap at each layer is equal to the out channels of the corresponding filter layer.
- At index 0, filter has 64 channels as output channels and the corresponding featuremap at 0^{th} index has 64 as the number of out channels. This infers that the corresponding filter & feature maps are pairs.

2.2 Visualizations

2.2.1 For three input images of different classes, show three pairs of filters and corresponding feature maps, each from a different layer in AlexNet. Indicate which layers you chose. For each pair, briefly explain what the filter is doing (for example: horizontal edge detection*)* which should be confirmed by the corresponding feature map. [15 marks]

Image #1, class: Power drill

	Filter	Feature map	Brief explanation

Early layer	**CONV_0**
	`torch.Size([64, 3, 11, 11])`
	1 of 64 filters of Channel0
	
		`img shape: torch.Size([1, 3, 300, 348])`
		`torch.Size([1, 64, 74, 86])`
		1 of 64 CONV2 Feature
		
		1 of 64 CONV2 Feature after Relu
		
			<u>Filter layer:</u>
			• Input channels=3
			• Output channels=64
			• To each input channels, 64 filter is applied
			• Filter is identifying the vertical lines
			
			<u>Feature Map:</u>
			• By applying 64 filters on each of the 3 input channel, we get 64 feature maps
			(image must pass through previous layers before reaching to this layer)

Intermediate layer
Layer 3
	CONV_6
	`torch.Size([384, 192, 3, 3])`
	1 of 384 filters of Channel0
	
		`img shape: torch.Size([1, 3, 300, 348])`
		`torch.Size([1, 384, 17, 20])`
		1 of 384 CONV2 Feature
		
		1 of 384 CONV2 Feature after Relu
		
			<u>Filter layer:</u>
			• Input channels=192
			• Output channels=384
			• To each input channels, 384 filter is applied
			
			<u>Feature Map:</u>
			• By applying 384 filters on each of the 192 input channel, we get 384 feature maps
			(image must pass through previous layers before reaching to this layer)

Deep layer	**CONV_10** `torch.Size([256, 256, 3, 3])`	`img shape: torch.Size([1, 3, 300, 348])` `torch.Size([1, 256, 17, 20])`	**<u>Filter layer:</u>** • Input channels=256 • Output channels=256

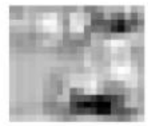
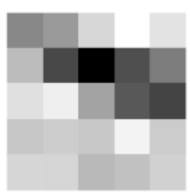
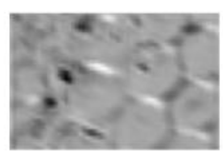
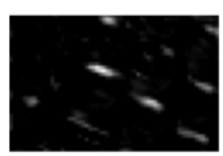
	1 of 256 filters of Channel0 	1 of 256 CONV2 Feature  1 of 256 CONV2 Feature after Relu 	• To each input channels, 256 filter is applied **<u>Feature Map:</u>** • By applying 256 filters on each of the 256 input channel, we get 256 feature maps (image must pass through previous layers before reaching to this layer)

Image #2, class: **Orange** (imagenet10/Alexnet/jybcucurxm.JPEG)

	Filter	Feature map	Brief explanation

Early layer	**CONV_3**
	`torch.Size([192, 64, 5, 5])`
	1 of 192 filters of Channel0
	
		`img shape: torch.Size([1, 3, 333, 500])`
		`torch.Size([1, 192, 40, 61])`
		1 of 192 CONV2 Feature
		
		1 of 192 CONV2 Feature after Relu
		
			<u>Filter layer:</u>
			• Input channels=64
			• Output channels=192
			• To each input channels, 192 filter is applied
			
			<u>Feature Map:</u>
			By applying 192 filters on each of the 64 input channel, we get 192 feature maps
			(image must pass through previous layers before reaching to this layer)

Intermediate layer	**CONV_8** `torch.Size([256, 384, 3, 3])` 1 of 256 filters of Channel0	1 of 256 CONV2 Feature	**<u>Filter layer:</u>** • Input channels=384 • Output channels=256

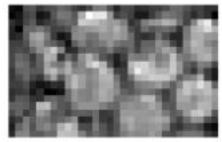
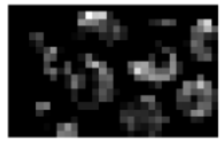
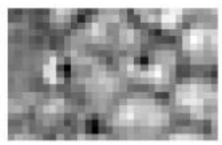
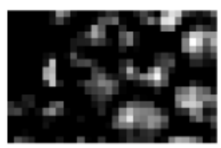
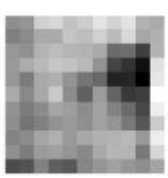
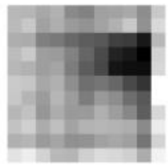
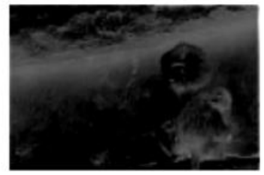
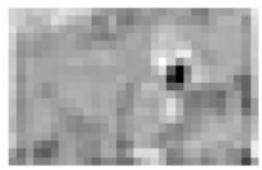
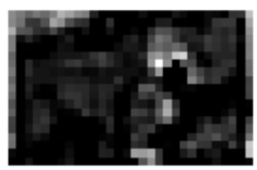
		 1 of 256 CONV2 Feature after Relu 	• To each input channels, 256 filter is applied **<u>Feature Map:</u>** By applying 256 filters on each of the 384 input channel, we get 256feature maps (image must pass through previous layers before reaching to this layer)
Deep layer	**CONV_10** `torch.Size([256, 256, 3, 3])` 	1 of 256 CONV2 Feature  1 of 256 CONV2 Feature after Relu 	**<u>Filter layer:</u>** • Input channels=256 • Output channels=256 • To each input channels, 256 filter is applied **<u>Feature Map:</u>** By applying 256 filters on each of the 256 input channel, we get 256 feature maps (image must pass through previous layers before reaching to this layer)

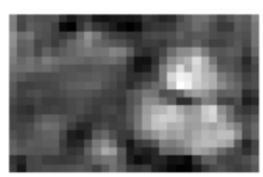
Image #3, class: Baboon

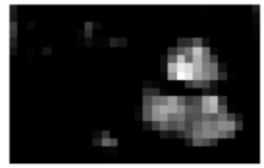
	Filter	Feature map	Brief explanation

Early layer	**CONV_0**
	`torch.Size([64, 3, 11, 11])`
	1 of 64 filters of Channel0
	
	1 of 64 filters of Channel1
		`img shape: torch.Size([1, 3, 333, 500])`
		`torch.Size([1, 64, 82, 124])`
		1 of 64 CONV2 Feature
		
			<u>Filter layer:</u>
			• Input channels=3
			• Output channels=64
			• To each input channels, 64 filter is applied
			• Filter is identifying low intensity dark region surrounded

		1 of 64 CONV2 Feature after Relu 	by high intensity region **<u>Feature Map:</u>** By applying 64 filters on each of the 3 input channel, we get 64 feature maps

Intermediate layer	**CONV_6**
	
	`torch.Size([384, 192, 3, 3])`
	
	1 of 192 filters of Channel0
	
		`img shape: torch.Size([1, 3, 333, 500])`
		`torch.Size([1, 384, 19, 30])`
		
		1 of 384 CONV2 Feature
		
		1 of 384 CONV2 Feature after Relu
		
			<u>Filter layer:</u>
			• Input channels=192
			• Output channels=384
			• To each input channels, 384 filter is applied
			
			<u>Feature Map:</u>
			By applying 384 filters on each of the 192 input channels of the image, we get 384 feature maps. (image must pass through previous layers before reaching to this layer)

Deep layer	**CONV_10**
	
	`torch.Size([256, 256, 3, 3])`
	1 of 256 filters of Channel0
	
		`img shape: torch.Size([1, 3, 333, 500])`
		`torch.Size([1, 256, 19, 30])`
		
		1 of 256 CONV2 Feature
		
		1 of 256 CONV2 Feature after RELU
			<u>Filter layer:</u>
			• Input channels=256
			• Output channels=256
			• To each input channels, 256 filter is applied
			
			<u>Feature Map:</u>
			By applying 256 filters on each of the 256 input channel of the image, we get 256 feature maps. (image must pass through

			previous layers before reaching to this layer),

2.2.2 Comment on how the filters and feature maps change with depth into the network. [5 marks]

- Deeper networks recognize complex features
- Initial layers detect edges, lines & corners. Using these base features, further layers can detect shapes and in later layers the shapes are used to detect complex features.
- In our filter & feature map example, we see that the 1st convolutional layer has filters that looked like lines or stripes in different directions. When an image is passed through this filter, the feature map generated has the edges & corners highlighted.
- Initial layer feature map has a higher image dimension and a smaller number of channels. In further layers, we increase the number of features(channels) and reduce the image dimension. When the image dimension is reduced, the prominent features of the image are retained as features by increasing the number of out channels.

Marks reserved for overall quality of report. [5 marks]

No response needed here.