Readme

- This project is done with the collaboration with Elvy Chen.
- The project is the course project of COGS 181: Neural Networks/Deep Learning. Here, we explore the ResNet18 Structure using the Cifar Dataset. We first extract some layers from ResNet18 without pretraining to be our baseline model for image classification on Cifar Dataset. Then, we perform transfer learning by utilizing the first 6 layers of ResNet18 and combine them with our own model architecture to develop another image classification model.
- In this notebook, we showed the proper steps of training a convolutional neural network from preprocessing the data to model evaluation.

Import packages

In [1]:

```
%matplotlib inline
import torchvision.datasets as datasets
from torchvision import models
import torch.utils.data as data
from torch.optim import lr_scheduler
import matplotlib.pyplot as plt
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pandas as pd
import numpy as np
import os
import platform
import time
import copy
import random
from torchvision import models
from util import *
from model import *
```

Load dataset

We load the Cifar dataset from pytorch and perform some preprocessing of data:

- randomly rotate images
- randomly flip the images
- Normalize the values of images

Then, we create train, validation, and test dataloaders so that we can load the data during the training process.

In [2]:

```
transform_1 = transforms.Compose(
    [transforms.RandomRotation(20),
     transforms.RandomHorizontalFlip(0.5),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset_1 = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform_1)


train_size = int(0.8 * len(trainset_1))
test_size = len(trainset_1) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(trainset_1, [train_si
ze, test_size])

trainloader_1 = torch.utils.data.DataLoader(train_dataset, batch_size=32,
                                          shuffle=True, num_workers=0, worker_in
it_fn=2)

valloader_1 = torch.utils.data.DataLoader(val_dataset, batch_size=32,
                                          shuffle=True, num_workers=0, worker_in
it_fn=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=32,
                                         shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Define parameters

We define more parameters to train the model and print out the ResNet18 structure that we defined.

In [3]:

```
model_1 = ResNet18()
#define loss function, optimizer, scheduler, batch size, and name of the model t
o be saved.
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model_1.parameters(), lr=0.001)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
batch_size = 32
save_path = "./ResNet18.pt"
print(model_1)
```

```
ResNet18(
  (features): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddin
g=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
    (relu): LeakyReLU(negative_slope=0.01, inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilatio
n=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bi
as=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
```

```
        )
      )
      (layer3): Sequential()
      (layer4): Sequential()
      (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
      (fc): Linear(in_features=128, out_features=10, bias=True)
    )
)
```

Training

We call the train_model function for training

In [4]:

```
model_resnet18,train_acc, train_loss,val_acc,val_loss = train_model(model_1, cri
terion, optimizer, exp_lr_scheduler,
                                                                    trainloader_1, val
loader_1, batch_size,
                                                                    device, save_path,
num_epochs=30)
```

```
Epoch 1/30
----------
Train Loss: 1.4463 Acc: 0.4715
Validation Loss: 1.2140 Acc: 0.5645
Validation loss decreased (inf --> 1.213967).  Saving model ...
Epoch 2/30
----------
Train Loss: 1.1364 Acc: 0.5956
Validation Loss: 1.0936 Acc: 0.6087
Validation loss decreased (1.213967 --> 1.093576).  Saving model ...
Epoch 3/30
----------
Train Loss: 0.9907 Acc: 0.6551
Validation Loss: 0.9744 Acc: 0.6586
Validation loss decreased (1.093576 --> 0.974401).  Saving model ...
Epoch 4/30
----------
Train Loss: 0.9107 Acc: 0.6810
Validation Loss: 0.8681 Acc: 0.6939
Validation loss decreased (0.974401 --> 0.868076).  Saving model ...
Epoch 5/30
----------
Train Loss: 0.8395 Acc: 0.7074
Validation Loss: 0.8333 Acc: 0.7071
Validation loss decreased (0.868076 --> 0.833295).  Saving model ...
Epoch 6/30
----------
Train Loss: 0.7182 Acc: 0.7514
Validation Loss: 0.7475 Acc: 0.7370
Validation loss decreased (0.833295 --> 0.747463).  Saving model ...
Epoch 7/30
----------
Train Loss: 0.6857 Acc: 0.7586
Validation Loss: 0.6953 Acc: 0.7552
Validation loss decreased (0.747463 --> 0.695264).  Saving model ...
Epoch 8/30
----------
Train Loss: 0.6498 Acc: 0.7706
Validation Loss: 0.7249 Acc: 0.7465
Epoch 9/30
----------
Train Loss: 0.6253 Acc: 0.7823
Validation Loss: 0.6833 Acc: 0.7575
Validation loss decreased (0.695264 --> 0.683334).  Saving model ...
Epoch 10/30
----------
Train Loss: 0.6015 Acc: 0.7892
Validation Loss: 0.6850 Acc: 0.7662
Epoch 11/30
----------
Train Loss: 0.5376 Acc: 0.8136
Validation Loss: 0.6383 Acc: 0.7796
Validation loss decreased (0.683334 --> 0.638291).  Saving model ...
Epoch 12/30
----------
Train Loss: 0.5092 Acc: 0.8215
Validation Loss: 0.6520 Acc: 0.7763
Epoch 13/30
----------
Train Loss: 0.5043 Acc: 0.8230
Validation Loss: 0.6336 Acc: 0.7783
```

```
Validation loss decreased (0.638291 --> 0.633607).  Saving model ...
Epoch 14/30
----------
Train Loss: 0.4884 Acc: 0.8273
Validation Loss: 0.6322 Acc: 0.7820
Validation loss decreased (0.633607 --> 0.632171).  Saving model ...
Epoch 15/30
----------
Train Loss: 0.4745 Acc: 0.8321
Validation Loss: 0.6639 Acc: 0.7782
Epoch 16/30
----------
Train Loss: 0.4439 Acc: 0.8454
Validation Loss: 0.6098 Acc: 0.7935
Validation loss decreased (0.632171 --> 0.609791).  Saving model ...
Epoch 17/30
----------
Train Loss: 0.4248 Acc: 0.8515
Validation Loss: 0.6143 Acc: 0.7923
Epoch 18/30
----------
Train Loss: 0.4147 Acc: 0.8560
Validation Loss: 0.6223 Acc: 0.7885
Epoch 19/30
----------
Train Loss: 0.4025 Acc: 0.8584
Validation Loss: 0.6256 Acc: 0.7892
Epoch 20/30
----------
Train Loss: 0.3984 Acc: 0.8619
Validation Loss: 0.6341 Acc: 0.7908
Epoch 21/30
----------
Train Loss: 0.3758 Acc: 0.8683
Validation Loss: 0.6133 Acc: 0.7956
Epoch 22/30
----------
Train Loss: 0.3764 Acc: 0.8684
Validation Loss: 0.6161 Acc: 0.7919
Epoch 23/30
----------
Train Loss: 0.3686 Acc: 0.8695
Validation Loss: 0.6215 Acc: 0.7943
Epoch 24/30
----------
Train Loss: 0.3687 Acc: 0.8698
Validation Loss: 0.6200 Acc: 0.7932
Epoch 25/30
----------
Train Loss: 0.3623 Acc: 0.8729
Validation Loss: 0.6175 Acc: 0.7962
Epoch 26/30
----------
Train Loss: 0.3504 Acc: 0.8780
Validation Loss: 0.6107 Acc: 0.7950
Epoch 27/30
----------
Train Loss: 0.3497 Acc: 0.8776
Validation Loss: 0.6273 Acc: 0.7915
Epoch 28/30
----------
```

```
Train Loss: 0.3440 Acc: 0.8790
Validation Loss: 0.6077 Acc: 0.7998
Validation loss decreased (0.609791 --> 0.607709).  Saving model ...
Epoch 29/30
----------
Train Loss: 0.3435 Acc: 0.8785
Validation Loss: 0.6249 Acc: 0.7949
Epoch 30/30
----------
Train Loss: 0.3435 Acc: 0.8807
Validation Loss: 0.6209 Acc: 0.8008
Training complete in 12m 32s
minimum val loss: 0.607709
```

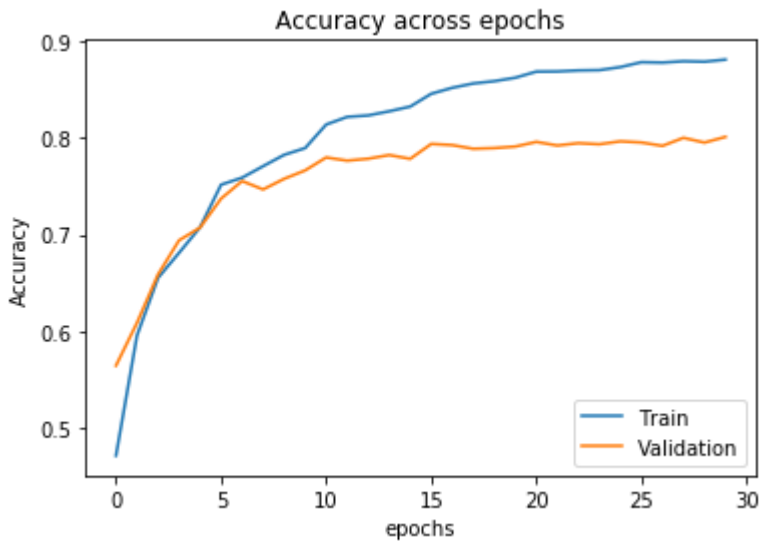
Visualizing the training and validation accuracy and loss

In [5]:

```
ax = plt.gca()
plt.plot(train_acc)
plt.plot(val_acc)
plt.title("Accuracy across epochs")
plt.xlabel("epochs")
plt.ylabel("Accuracy")
ax.legend(['Train','Validation'],loc='lower right')
```

Out[5]:

```
<matplotlib.legend.Legend at 0x7f22acfe98d0>
```

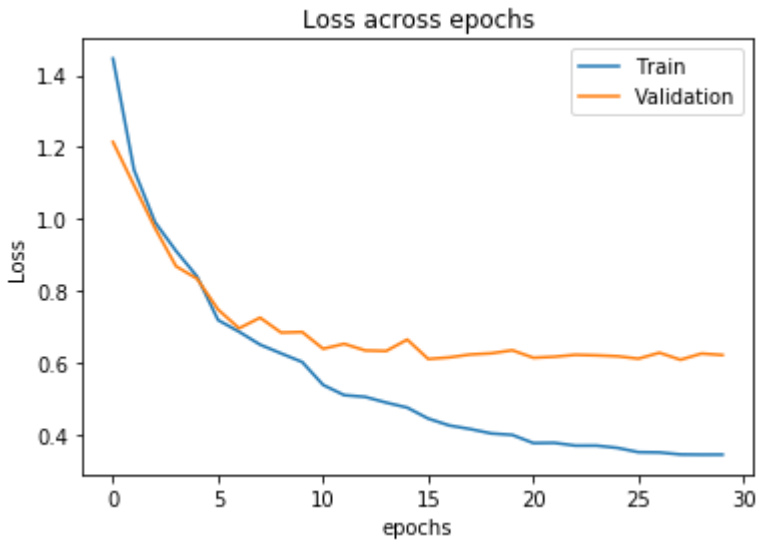


In [6]:

```
ax = plt.gca()
plt.plot(train_loss)
plt.plot(val_loss)
plt.title("Loss across epochs")
plt.xlabel("epochs")
plt.ylabel("Loss")
ax.legend(['Train','Validation'],loc='upper right')
```

Out[6]:

```
<matplotlib.legend.Legend at 0x7f22a417edd8>
```



Evaluation

- Calculate the overall accuracy of all class prediction
- Calculate the accuracy of every specific class prediction

In [7]:

```
calculate_accuracy(model_resnet18, trainloader_1, "train")
print("-" * 50)
calculate_accuracy(model_resnet18, valloader_1, "validation")
print("-" * 50)
calculate_accuracy(model_resnet18, testloader, "test")
```

```
Accuracy of the network all images in train set: 89.90
--------------------------------------------------
Accuracy of the network all images in validation set: 79.60
--------------------------------------------------
Accuracy of the network all images in test set: 81.19
```

In [8]:

```
calculate_accuracy_specific(model_resnet18, trainloader_1, classes, "train")
print("-" * 50)
calculate_accuracy_specific(model_resnet18, valloader_1, classes, "validation")
print("-" * 50)
calculate_accuracy_specific(model_resnet18, testloader, classes, "tetestst")
```

```
Accuracy of plane in train set: 92.13
Accuracy of car in train set: 93.17
Accuracy of bird in train set: 87.63
Accuracy of cat in train set: 75.39
Accuracy of deer in train set: 91.07
Accuracy of dog in train set: 80.08
Accuracy of frog in train set: 90.91
Accuracy of horse in train set: 91.35
Accuracy of ship in train set: 94.71
Accuracy of truck in train set: 92.43
--------------------------------------------------
Accuracy of plane in validation set: 80.47
Accuracy of car in validation set: 95.28
Accuracy of bird in validation set: 72.73
Accuracy of cat in validation set: 59.05
Accuracy of deer in validation set: 76.38
Accuracy of dog in validation set: 60.47
Accuracy of frog in validation set: 84.43
Accuracy of horse in validation set: 84.73
Accuracy of ship in validation set: 89.15
Accuracy of truck in validation set: 88.72
--------------------------------------------------
Accuracy of plane in tetestst set: 84.07
Accuracy of car in tetestst set: 86.09
Accuracy of bird in tetestst set: 69.74
Accuracy of cat in tetestst set: 62.79
Accuracy of deer in tetestst set: 79.55
Accuracy of dog in tetestst set: 70.80
Accuracy of frog in tetestst set: 82.64
Accuracy of horse in tetestst set: 89.84
Accuracy of ship in tetestst set: 87.61
Accuracy of truck in tetestst set: 89.71
```

Transfer Learning with resnet18

Now, we utilize the first six layers from resnet18 and fix its gradient. Then, we define additional layers to build another model

Define parameters

We define more parameters to train the model and print out the ResNet18_tr structure that we defined.

In [9]:

```
model_2 = ResNet18_tr()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model_2.parameters(), lr=0.001)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
batch_size = 32
save_path = "./ResNet18_tr"
print(model_2)
```

```
ResNet18_tr(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=
(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, c
eil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bi
as=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_stats=True)
```

```
        )
      )
    )
    (conv1): Conv2d(128, 256, kernel_size=(2, 2), stride=(1, 1), paddi
ng=(1, 1))
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1))
    (bh1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
    (pool3): AvgPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
    (lin1): Linear(in_features=1024, out_features=500, bias=True)
    (lin2): Linear(in_features=500, out_features=10, bias=True)
)
```

Training

We call the train_model function for training

In [10]:

```
model_resnet18_tr,train_acc, train_loss,val_acc,val_loss = train_model(model_2,
criterion, optimizer,
                                                                       exp_lr_sc
heduler,trainloader_1, valloader_1,
                                                                       batch_siz
e,device, save_path, num_epochs=30)
```

```
Epoch 1/30
----------
Train Loss: 1.0969 Acc: 0.6110
Validation Loss: 0.9049 Acc: 0.6814
Validation loss decreased (inf --> 0.904923).  Saving model ...
Epoch 2/30
----------
Train Loss: 0.9152 Acc: 0.6787
Validation Loss: 0.8393 Acc: 0.7033
Validation loss decreased (0.904923 --> 0.839304).  Saving model ...
Epoch 3/30
----------
Train Loss: 0.8418 Acc: 0.7038
Validation Loss: 0.8440 Acc: 0.7010
Epoch 4/30
----------
Train Loss: 0.7937 Acc: 0.7222
Validation Loss: 0.8209 Acc: 0.7219
Validation loss decreased (0.839304 --> 0.820897).  Saving model ...
Epoch 5/30
----------
Train Loss: 0.7596 Acc: 0.7357
Validation Loss: 0.7664 Acc: 0.7306
Validation loss decreased (0.820897 --> 0.766399).  Saving model ...
Epoch 6/30
----------
Train Loss: 0.6738 Acc: 0.7615
Validation Loss: 0.7172 Acc: 0.7441
Validation loss decreased (0.766399 --> 0.717173).  Saving model ...
Epoch 7/30
----------
Train Loss: 0.6396 Acc: 0.7763
Validation Loss: 0.6973 Acc: 0.7575
Validation loss decreased (0.717173 --> 0.697298).  Saving model ...
Epoch 8/30
----------
Train Loss: 0.6235 Acc: 0.7803
Validation Loss: 0.6968 Acc: 0.7514
Validation loss decreased (0.697298 --> 0.696834).  Saving model ...
Epoch 9/30
----------
Train Loss: 0.6031 Acc: 0.7887
Validation Loss: 0.6857 Acc: 0.7648
Validation loss decreased (0.696834 --> 0.685728).  Saving model ...
Epoch 10/30
----------
Train Loss: 0.5940 Acc: 0.7917
Validation Loss: 0.6778 Acc: 0.7613
Validation loss decreased (0.685728 --> 0.677830).  Saving model ...
Epoch 11/30
----------
Train Loss: 0.5426 Acc: 0.8054
Validation Loss: 0.6464 Acc: 0.7779
Validation loss decreased (0.677830 --> 0.646357).  Saving model ...
Epoch 12/30
----------
Train Loss: 0.5284 Acc: 0.8148
Validation Loss: 0.6301 Acc: 0.7771
Validation loss decreased (0.646357 --> 0.630071).  Saving model ...
Epoch 13/30
----------
```

```
Train Loss: 0.5147 Acc: 0.8192
Validation Loss: 0.6534 Acc: 0.7760
Epoch 14/30
----------
Train Loss: 0.5094 Acc: 0.8195
Validation Loss: 0.6408 Acc: 0.7753
Epoch 15/30
----------
Train Loss: 0.5003 Acc: 0.8236
Validation Loss: 0.6589 Acc: 0.7754
Epoch 16/30
----------
Train Loss: 0.4711 Acc: 0.8345
Validation Loss: 0.6438 Acc: 0.7785
Epoch 17/30
----------
Train Loss: 0.4612 Acc: 0.8350
Validation Loss: 0.6299 Acc: 0.7853
Validation loss decreased (0.630071 --> 0.629945).  Saving model ...
Epoch 18/30
----------
Train Loss: 0.4548 Acc: 0.8403
Validation Loss: 0.6525 Acc: 0.7795
Epoch 19/30
----------
Train Loss: 0.4492 Acc: 0.8403
Validation Loss: 0.6366 Acc: 0.7850
Epoch 20/30
----------
Train Loss: 0.4472 Acc: 0.8408
Validation Loss: 0.6526 Acc: 0.7768
Epoch 21/30
----------
Train Loss: 0.4291 Acc: 0.8478
Validation Loss: 0.6352 Acc: 0.7864
Epoch 22/30
----------
Train Loss: 0.4287 Acc: 0.8481
Validation Loss: 0.6393 Acc: 0.7889
Epoch 23/30
----------
Train Loss: 0.4232 Acc: 0.8503
Validation Loss: 0.6388 Acc: 0.7834
Epoch 24/30
----------
Train Loss: 0.4228 Acc: 0.8500
Validation Loss: 0.6321 Acc: 0.7905
Epoch 25/30
----------
Train Loss: 0.4180 Acc: 0.8510
Validation Loss: 0.6372 Acc: 0.7798
Epoch 26/30
----------
Train Loss: 0.4072 Acc: 0.8550
Validation Loss: 0.6293 Acc: 0.7872
Validation loss decreased (0.629945 --> 0.629293).  Saving model ...
Epoch 27/30
----------
Train Loss: 0.4073 Acc: 0.8555
Validation Loss: 0.6381 Acc: 0.7836
Epoch 28/30
```

```
----------
Train Loss: 0.4066 Acc: 0.8554
Validation Loss: 0.6354 Acc: 0.7867
Epoch 29/30
----------
Train Loss: 0.4034 Acc: 0.8572
Validation Loss: 0.6417 Acc: 0.7811
Epoch 30/30
----------
Train Loss: 0.4048 Acc: 0.8584
Validation Loss: 0.6251 Acc: 0.7893
Validation loss decreased (0.629293 --> 0.625080).  Saving model ...
Training complete in 7m 3s
minimum val loss: 0.625080
```

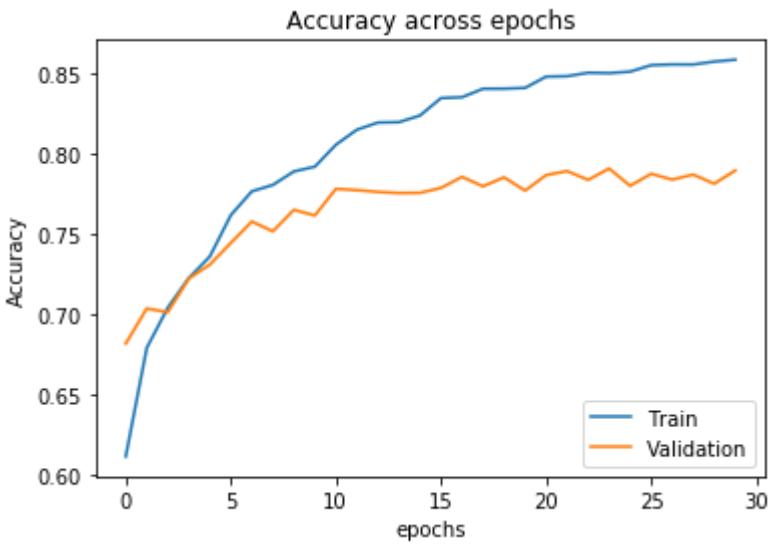
Visualizing the training and validation accuracy and loss

In [11]:

```
ax = plt.gca()
plt.plot(train_acc)
plt.plot(val_acc)
plt.title("Accuracy across epochs")
plt.xlabel("epochs")
plt.ylabel("Accuracy")
ax.legend(['Train','Validation'],loc='lower right')
```

Out[11]:

```
<matplotlib.legend.Legend at 0x7f22a4031f28>
```

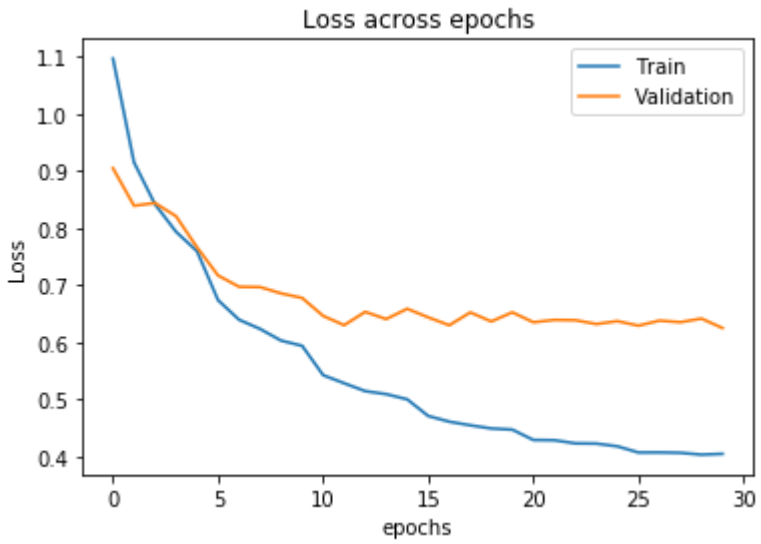


In [12]:

```
ax = plt.gca()
plt.plot(train_loss)
plt.plot(val_loss)
plt.title("Loss across epochs")
plt.xlabel("epochs")
plt.ylabel("Loss")
ax.legend(['Train','Validation'],loc='upper right')
```

Out[12]:

```
<matplotlib.legend.Legend at 0x7f22a0f99e48>
```



Evaluation

- Calculate the overall accuracy of all class prediction
- Calculate the accuracy of every specific class prediction

In [13]:

```
calculate_accuracy(model_resnet18_tr, trainloader_1, "train")
print("-" * 50)
calculate_accuracy(model_resnet18_tr, valloader_1, "validation")
print("-" * 50)
calculate_accuracy(model_resnet18_tr, testloader, "test")
```

```
Accuracy of the network all images in train set: 86.85
--------------------------------------------------
Accuracy of the network all images in validation set: 79.00
--------------------------------------------------
Accuracy of the network all images in test set: 81.10
```

In [14]:

```
calculate_accuracy_specific(model_resnet18_tr, trainloader_1, classes, "train")
print("-" * 50)
calculate_accuracy_specific(model_resnet18_tr, valloader_1, classes, "validatio
n")
print("-" * 50)
calculate_accuracy_specific(model_resnet18_tr, testloader, classes, "teteststst")
```

```
Accuracy of plane in train set: 88.43
Accuracy of car in train set: 91.41
Accuracy of bird in train set: 84.90
Accuracy of cat in train set: 73.96
Accuracy of deer in train set: 85.86
Accuracy of dog in train set: 81.22
Accuracy of frog in train set: 89.17
Accuracy of horse in train set: 87.76
Accuracy of ship in train set: 93.45
Accuracy of truck in train set: 90.68
--------------------------------------------------
Accuracy of plane in validation set: 85.25
Accuracy of car in validation set: 88.43
Accuracy of bird in validation set: 72.80
Accuracy of cat in validation set: 57.52
Accuracy of deer in validation set: 79.39
Accuracy of dog in validation set: 64.46
Accuracy of frog in validation set: 83.33
Accuracy of horse in validation set: 89.76
Accuracy of ship in validation set: 86.09
Accuracy of truck in validation set: 83.47
--------------------------------------------------
Accuracy of plane in teteststst set: 79.65
Accuracy of car in teteststst set: 90.43
Accuracy of bird in teteststst set: 78.95
Accuracy of cat in teteststst set: 69.77
Accuracy of deer in teteststst set: 76.52
Accuracy of dog in teteststst set: 68.14
Accuracy of frog in teteststst set: 85.95
Accuracy of horse in teteststst set: 84.38
Accuracy of ship in teteststst set: 86.73
Accuracy of truck in teteststst set: 89.71
```

Conclusion

Based on the result, the ResNet18 model outperforms the ResNet18_tr model. This is a bit counterintuitive because originally we thought that transfer learning would help the model to converge faster and achieve higher classification accuracy and lower loss. One reason that we think might be the cause is the size of the Cifar dataset. Cifar dataset is relatively small compared to the imagenet that was used by the originally resnet18 model. Therefore, with only ten classes, the feature extraction from resnet18 might not be well suited for Cifar dataset prediction because some learned features might not be applicable to the Cifar dataset.

In [ ]:

```

```