In [ ]:

```python
from __future__ import print_function, division
import matplotlib.pyplot as plt
import numpy as np
# import tensorflow as tf
# import tensorflow.keras as keras
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision

from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

# plt.ion()   # interactive mode
```

In [ ]:

```python
data_transforms = {
    'train': transforms.Compose([
        # transforms.RandomResizedCrop(224),
        # transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.20
    ]),
    'val': transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.20
    ]),
}



image_datasets = {'train': datasets.CIFAR100('data',train=True,
                                          transform=data_transforms['train']
                  ,'val': datasets.CIFAR100('data',train=False,
                                          transform=data_transforms['val'],d
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=
                                             shuffle=True)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

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

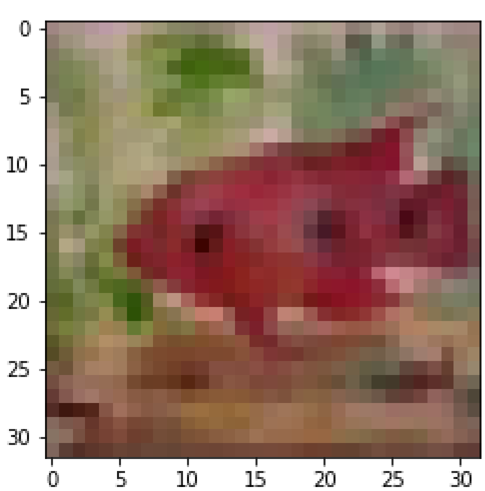
CIFAR-100 Dataset:

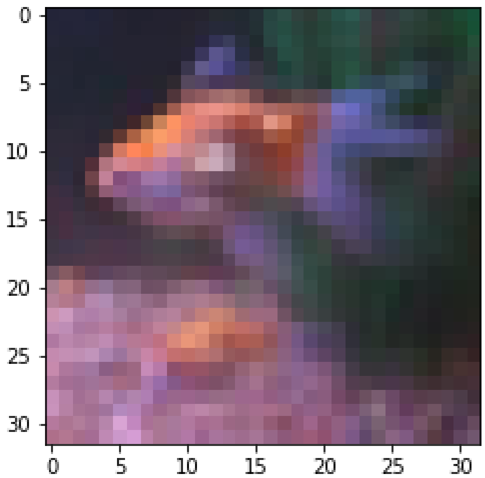
No of classes = 100 \ No of images from each class = 600 \

In [ ]:

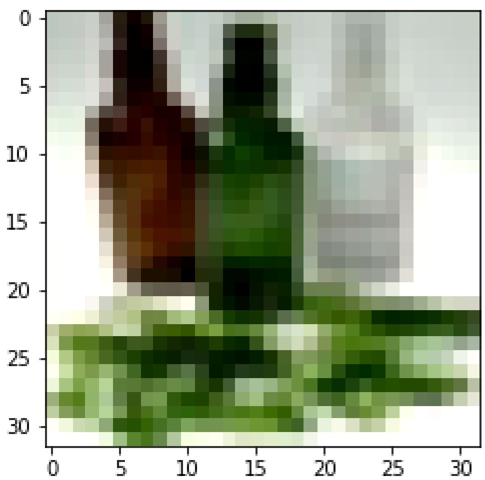
```python
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)


append=[]
sqind = []
while(len(append)<2 or len(sqind)<2):
    inputs, classes = next(iter(dataloaders['train']))
    for i in range(len(classes)):
        if(len(append)==2 and len(sqind)==2):
            break
        if(classes[i]==1 and len(append)<2):
            append.append(inputs[i])
        if(classes[i]==9 and len(sqind)<2):
            sqind.append(inputs[i])
print(class_names[1])
imshow(append[0])
imshow(append[1])
plt.show()
print(class_names[9])
imshow(sqind[0])
plt.show()
imshow(sqind[1])
plt.show()
print(len(dataloaders['train']),len(dataloaders['val']))
```

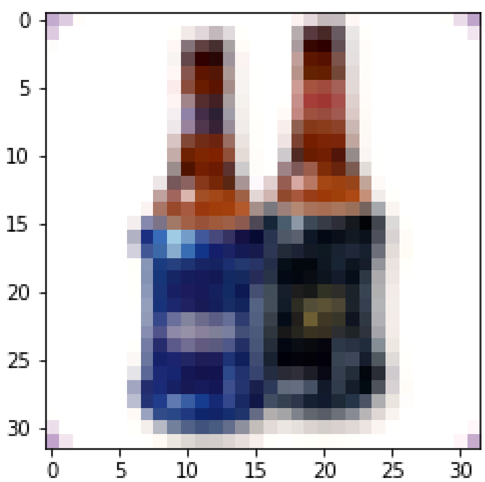
```
aquarium_fish
```





```
bottle
```





```
782 157
```

In [ ]:
```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    acc = []
    for epoch in range(num_epochs):
        # print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        # print('-' * 10)
        epoch_time = time.time()
        # Each epoch has a training and validation phase
        for phase in ['train','val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
```

```
            acc.append(epoch_acc.item())
#             print(acc)
            # if(epoch>10):
            #     if(np.mean(acc[-10:])>epoch_acc):
            #         break

        # if(epoch>10):
        #         if(np.mean(acc[-10:])>epoch_acc):
        #             break
        print('Epoch time ',time.time()-epoch_time)
        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

In [ ]:

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('predicted: {}'.format(class_names[preds[j]]))
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

In [ ]:
```python
# 1.b
model_ft = models.resnet50(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_name
model_ft.fc = nn.Linear(num_ftrs, 100)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=20, gamma=0.1
```

In [ ]:
```python
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=60)
```

```
train Loss: 0.1670 Acc: 0.9500
val Loss: 1.9103 Acc: 0.5982
Epoch time  58.50404477119446

train Loss: 0.1301 Acc: 0.9613
val Loss: 1.9514 Acc: 0.5944
Epoch time  57.047507524490356

train Loss: 0.1153 Acc: 0.9658
val Loss: 1.9796 Acc: 0.6012
Epoch time  57.02501702308655

train Loss: 0.0951 Acc: 0.9720
val Loss: 2.0204 Acc: 0.5990
Epoch time  57.272584438323975

train Loss: 0.0932 Acc: 0.9724
val Loss: 2.0502 Acc: 0.5947
Epoch time  57.72999143600464
```

In [ ]:
```python

```

In [ ]:
```
# 1.c
model_ft = models.resnet50(pretrained=True)
num_ftrs = model_ft.fc.in_features

model_ft.fc = nn.Linear(num_ftrs, 100)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.01, momentum=0.9)

# Choosing large step_size so that the learning rate is not changed
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10000, gamma=
```

In [ ]:
```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=200)
```

```
val Loss: 2.8747 Acc: 0.5826
Epoch time  58.596367597579956

train Loss: 0.0064 Acc: 0.9983
val Loss: 2.8930 Acc: 0.5801
Epoch time  57.54751014709473

train Loss: 0.0054 Acc: 0.9985
val Loss: 3.0666 Acc: 0.5672
Epoch time  56.887218713760376

train Loss: 0.0052 Acc: 0.9985
val Loss: 2.9798 Acc: 0.5787
Epoch time  56.546945571899414

train Loss: 0.0042 Acc: 0.9990
val Loss: 2.9950 Acc: 0.5794
Epoch time  57.32757496833801
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

```
model_ft = models.resnet50(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_name
model_ft.fc = nn.Linear(num_ftrs, 100)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.1, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10000, gamma=
```

In [ ]:

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=200)
```

```
train Loss: 0.0745 Acc: 0.9752
val Loss: 5.7817 Acc: 0.3835
Epoch time  30.57904863357544

train Loss: 0.0691 Acc: 0.9780
val Loss: 5.8785 Acc: 0.3758
Epoch time  30.87091326713562

train Loss: 0.0616 Acc: 0.9800
val Loss: 5.7691 Acc: 0.3872
Epoch time  30.638628244400024

train Loss: 0.0663 Acc: 0.9787
val Loss: 5.8226 Acc: 0.3886
Epoch time  31.256396532058716

Training complete in 25m 52s
Best val Acc: 0.388600
```

All the three learning rates give very high training accuracies. \ The first learning rate used gives the highest validation accuracy on the target set. Validation acc = 0.6231 \ The second learning rate gives a lower accuracy. Validation acc = 0.5794 \ And the third onne gives the least among the three. Validation acc = 0.0.3886 \ Therefore the learning rate which changes from 0.0001 to 0.000001 gives the best accuracy

In [ ]:

```

```

In [ ]:

```

```

In [ ]:
```
# 2
model_conv = torchvision.models.resnet50(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 100)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

```

In [ ]:
```

```

In [ ]:
```
# 2.a
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=1, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=10000, gamm
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=200)
```

```
train Loss: 142.7233 Acc: 0.1478
val Loss: 171.0052 Acc: 0.1797
Epoch time  18.774144172668457

train Loss: 141.3311 Acc: 0.1995
val Loss: 151.4047 Acc: 0.2069
Epoch time  19.278165340423584

train Loss: 138.1913 Acc: 0.2194
val Loss: 159.6138 Acc: 0.2148
Epoch time  23.403469800949097

train Loss: 136.6077 Acc: 0.2316
val Loss: 171.3207 Acc: 0.2133
Epoch time  19.571969509124756

train Loss: 136.5718 Acc: 0.2446
val Loss: 160.7145 Acc: 0.2236
Epoch time  42.624640703201294
```

In [ ]:

```
model_conv = torchvision.models.resnet50(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 100)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.1, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=10000, gamm
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=200)
```

```
Epoch time  18.200000047683716

train Loss: 12.8065 Acc: 0.3433
val Loss: 18.7889 Acc: 0.2570
Epoch time  18.594185829162598

train Loss: 12.8265 Acc: 0.3414
val Loss: 20.6364 Acc: 0.2526
Epoch time  18.247527360916138

train Loss: 12.9073 Acc: 0.3421
val Loss: 19.8451 Acc: 0.2527
Epoch time  18.78353762626648

train Loss: 12.7418 Acc: 0.3437
val Loss: 18.2947 Acc: 0.2527
Epoch time  18.217689990997314

Training complete in 23m 8s
Best val Acc: 0.260000
```

In [ ]:

```
model_conv = torchvision.models.resnet50(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 100)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.01, momentum=0.9

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=10000, gamm
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=200)
```

```
Epoch time  18.62247633934021

train Loss: 2.3275 Acc: 0.4275
val Loss: 3.4606 Acc: 0.3033
Epoch time  18.950173139572144

train Loss: 2.3305 Acc: 0.4273
val Loss: 3.4212 Acc: 0.3136
Epoch time  18.578193426132202

train Loss: 2.3288 Acc: 0.4259
val Loss: 3.5612 Acc: 0.3058
Epoch time  18.40757417678833

train Loss: 2.3154 Acc: 0.4300
val Loss: 3.4103 Acc: 0.3092
Epoch time  18.22861409187317

Training complete in 15m 33s
Best val Acc: 0.317500
```

In [ ]:

```
model_conv = torchvision.models.resnet50(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 100)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=10000, gamm
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=200)
```

```
val Loss: 2.7751 Acc: 0.3316
Epoch time  19.38201332092285

train Loss: 2.3790 Acc: 0.4015
val Loss: 2.7780 Acc: 0.3347
Epoch time  18.83073139190674

train Loss: 2.3702 Acc: 0.4085
val Loss: 2.8138 Acc: 0.3345
Epoch time  18.913022756576538

train Loss: 2.3622 Acc: 0.4093
val Loss: 2.7819 Acc: 0.3343
Epoch time  18.922974348068237

train Loss: 2.3659 Acc: 0.4080
val Loss: 2.7631 Acc: 0.3355
Epoch time  18.783432960510254
```

In [ ]:

```

```

2.(a) The 0.001 learning rate gives the best accuracy on the target dataset = 0.3355

2.(b) The finetuning approach gave better results than the feature extraction approach. Among all the models, the finetuning model with a small learning rate ranging from 0.0001 to 0.000001 gave the best accuracy. This is because we expect the pretrained weights to be relatively good estimates and high learning rates may distort them too quickly. The feature extractor model may have given low accuracies because the ImageNet dataset which it is trained on might not be that similar to the CIFAR-100 dataset. So the feature extractioin is not as good as expected. As the resnet model has batch normalization layers, which might be input dependent, changing the input distribution drastically may affect the results.

In [ ]:
```
1
```

Problem 2

1. In the paper which uses weakly supervised learning, they use noisy labels(hashtags) to improve the accuracy.

Whereas the semi-supervised learninng paper uses a teacher-student model in which a teacher model is trained using labeled data and is used to label unseen unlabeled data.

2.(a) Yes, the models trained using hashtags are robust again noise labels. In the paper they pre-trained a ResNeXt model with 1B images and 17k labels where p% of the labels were randomly replaced with noise. p = 10% decreased the top-1 accuracy on ImageNet by only around 1%, and p = 25% decreased accuracy by about 2%.

(b) Hashtags follow Zipfian distribution and it may reduce the impact of some of the classes on the overall training loss. Resampling the hashtag distribution ensures that all classes are included in training. Resampling of the hashtag distribution is important in order to obtain good transfer to ImageNet image-classification tasks. Using uniform or square-root sampling leads to an accuracy improvement of 5 to 6% irrespective of the number of ImageNet classes in the transfer task.

3.a) The goal is to improve performance using unlabeled data. This paper suggest using two models, teacher and student so that one model can benefit from the other. The teacher model is trained using labeled data. The unlabeled data is then passed through the teacher model and top-K labels from each target variable are selected from this to create a new labeled dataset. The student model is then trained on this new dataset after which it is finetuned on the previvous labeled dataset. So the student model uses the labels generated from the teacher model to train itself.

Distillation is a procedure used to compress a large model into a smaller one. In distillation the teacher model makes prediction on unlabelled data, and the inferred labels are used to train the student in a supervised fashion which is very similar to the model suggested in the paper. Therefore the teacher-student model is a type of distillation technique.

The teacher and student model is needed since the teacher model selects the top-K images in the unlabeled dataset and this dataset allows us generate a new training set. Since the student model is always 'smaller' than the teacher model, it is a distillation technique.

b) K is the number of examples that are selected from the unlabeled dataset U for each target label. P corresponds to the number of relevant classes of an image. The reason for choosing P > 1 is that it is difficult to identify accurately under-represented objects, or some may be cut-off by more prominent co-occurring objects.

c) The new labeled dataset is created by selecting the top-K images from each class from the predictions of the teacher model. Yes an image from this dataset can belong to more than one class. As P>1, if the scores of the image for more than one class lies in the top K scores of those respectiive classes then it will belong to all such classes.

d) Increasing K initially increases the amount of data present for the student model and thus causes the accuracy to increase. But upon increasing K further, we observe a drop ini accuracy because this causes incorrect results from the output of the teacher model to be added to the new labeled data. As this increases the noise in the labels of the new dataset the accuracy decreases.

Problem 3

1. Achieving peak FLOPS requires customized libraries with intimate knowledge of the underlying hardware. Even specially tuned libraries may fall short of peak execution by as much as 40%. Instead of trying to measure and capture every source of inefficiency in every learning framework, the paper suggests taking a small number of representative deep learning workloads which contain convolutions, pooling, dropout, and fully connected layers and run them for a short time on a single GPU. Given observed total throughput and estimated total throughput on this benchmark, fit a scaling constant to estimate a platform percent of peak (PPP) parameter which captures the average relative inefficiency of the platform compared to peak FLOPS.

2. The VGG19 model has a conv3-256 and 2 conv3-512 layers in addition to the VGG16 moodel. Therefore additional FLOPs = 4,161,798,144 = 4162M

The distribution of FLOPs for the VGG16 model is as follows-

CONV:15360M, POOL:6M, ReLU:14M, FC:124M,

Upon adding the additional FLOPs,

The CONV layers FLOPs of the VGG19 network: 19522M

And the total FLOPs of the VGG19 network is: 15503M + 4162M = 19,665M

So the fraction of the total FLOPs attributed by convolution layers 0.9932.

3. The measured time and sum of layerwise timing for forward pass did not match on GPUs. This is because CUDA allows asynchronous programming. Before the time is measured, an API is called to ensure that all cores have finished their tasks. This synchronization before measuring time on the GPUs results in an extra overhead. Thererefore, the sum of layerwise timing on GPUs is longer than a full forward pass.

In a full forward pass, timing is only recorded at the last layer. Therefore, a core may be assigned with the computation of following layers and thus it can continuously perform the computation without synchronization. For example, after finishing the multiply-add operations for the matrix multiplication at a CONV layer, a core can continue to calculate the max function of next ReLU layer on the output of multiply-add operations. If layerwise timing is recorded, all cores have to wait until all multiply-add operations of the CONV layer have been completed.

To mitigate the overhead, they keep GPUs iteratively running the process in a way that GPU cores can continuously perform multiply-add operations without synchronization, before recording the end time. Then, the measurement overhead is amortized over all the iterations, giving accurate timing estimates. When the number of iterations is large the measurement overhead becomes insignificant.

4. NVidia Tesla K80: double PPP = 1.87 Tflops. Forward pass on VGG requires 15503M FLOPs; as such, one forward pass on a K80 would take (15503 x 10^6) / (1.87 x 10^12) = 0.00829037433s so throughput is 120 images/sec.

GoogLeNet: Inference time: (1606 x 10^6) / (1.87 x 10^12) = 0.0008588s/image. Throughoutput: 1164 images/sec.

ResNet: Inference time: (3922 x 10^6) / (1.87 x 10^12) = 0.0020973s/img. Throughput: 476 images/sec.

In [ ]:

```
1
```

Problem 4

Collaborated with Atul Balaji ab5246

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init
import torchvision
from torch.autograd import Variable

__all__ = ['resnet18', 'resnet20', 'resnet32', 'resnet44', 'resnet56', 'resn

def _weights_init(m):
    classname = m.__class__.__name__
    #print(classname)
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        init.kaiming_normal_(m.weight)

class LambdaLayer(nn.Module):
    def __init__(self, lambd):
        super(LambdaLayer, self).__init__()
        self.lambd = lambd

    def forward(self, x):
        return self.lambd(x)


class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1, option='A'):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stri
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padc
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != planes:
            if option == 'A':
                """
                For CIFAR10 ResNet paper uses option A.
                """
                self.shortcut = LambdaLayer(lambda x:
                                            F.pad(x[:, :, ::2, ::2], (0, 0,
            elif option == 'B':
                self.shortcut = nn.Sequential(
                     nn.Conv2d(in_planes, self.expansion * planes, kernel_si
                     nn.BatchNorm2d(self.expansion * planes)
                )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out


```

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 16

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1, bi
        self.bn1 = nn.BatchNorm2d(16)
        self.layer1 = self._make_layer(block, 16, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 32, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 64, num_blocks[2], stride=2)
        self.linear = nn.Linear(64, num_classes)

        self.apply(_weights_init)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion

        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = F.avg_pool2d(out, out.size()[3])
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out

def resnet18():
    # return torchvision.models.resnet18()
    return ResNet(BasicBlock, [3,3,2])

def resnet20():
    return ResNet(BasicBlock, [3, 3, 3])


def resnet32():
    return ResNet(BasicBlock, [5, 5, 5])


def resnet44():
    return ResNet(BasicBlock, [7, 7, 7])


def resnet56():
    return ResNet(BasicBlock, [9, 9, 9])

def resnet50():
    return ResNet(BasicBlock, [8, 8, 8])

def test(net):
    import numpy as np
```

```
    total_params = 0

    for x in filter(lambda p: p.requires_grad, net.parameters()):
        total_params += np.prod(x.data.numpy().shape)
    print("Total number of params", total_params)
    print("Total layers", len(list(filter(lambda p: p.requires_grad and len(



for net_name in __all__:
    if net_name.startswith('resnet'):
        print(net_name)
        test(globals()[net_name]())
        print()

```

```
resnet18
Total number of params 195738
Total layers 18

resnet20
Total number of params 269722
Total layers 20

resnet32
Total number of params 464154
Total layers 32

resnet44
Total number of params 658586
Total layers 44

resnet56
Total number of params 853018
Total layers 56

resnet50
Total number of params 755802
Total layers 50
```

In [2]:

```
import torch
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

In [3]:

```python
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np


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

batch_size = 128

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True)

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

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
```

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


```
horse  ship   dog  frog  deer  ship horse   dog truck plane plane truck  frog
car truck plane   car plane   cat   dog  deer  ship   cat   car   car   cat  de
er   dog  frog  deer  deer  frog plane plane truck  deer plane horse   car hors
e  bird  frog horse  ship   car  frog plane   cat  ship plane   cat  frog truck
ship   dog  frog  ship  frog  frog  bird   car truck   dog   dog   dog  ship
cat  deer truck   car horse  ship  ship  deer  bird   cat  deer   cat plane  de
er   car   dog  ship  bird  deer  ship  frog   dog   cat   dog   dog   dog  shi
p truck  ship  bird   cat   car  bird   dog truck  ship   cat truck truck   car
truck  ship horse   cat   dog  bird horse   cat plane   cat  deer   car   cat t
ruck  deer  frog truck plane  frog plane   dog   car
```

In [6]:
```
import time
import torch.optim as optim
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet18().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses1 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses1.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    print('epoch time',time.time()-s)
print('Finished Training')
```

```
y", line 1328, in __del__
    self._shutdown_workers()
  File "/opt/conda/lib/python3.7/site-packages/torch/utils/data/dataloader.p
y", line 1320, in _shutdown_workers
    if w.is_alive():
  File "/opt/conda/lib/python3.7/multiprocessing/process.py", line 151, in i
s_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process

(292, 2.1838934421539307, 0.004304968754342776)
(292, 4.227958679199219, 0.003536822247005315)
(292, 6.244589805603027, 0.007616428266355425)
(292, 8.25268292427063, 0.009005396787257751)
(292, 10.308573961257935, 0.016861269143600092)
(292, 12.375986099243164, 0.00898786271656198)
(292, 14.448589563369751, 0.006666903046663015)
epoch time 16.18087601661682

Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x
```

In [20]:
```
textfile = open("loss1.txt", "w")
for element in losses1:
    textfile.write(str(element) + "\n")
textfile.close()
```

In [12]:
```
import time
import torch.optim as optim
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet20().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses2 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses2.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    print('epoch time',time.time()-s)
print('Finished Training')
```

```
epoch time 14.947744369506836
(348, 1.929938793182373, 0.0011401226502315768)
(348, 3.8538639545440674, 0.0011999636884940294)
(348, 5.774469614028931, 0.0010624661568281412)
(348, 7.703281402587891, 0.001675010332124954)
(348, 9.6322660446167, 0.0026279223972351805)
(348, 11.554770946502686, 0.0022304148843264853)
(348, 13.482335567474365, 0.001752686094046019)
epoch time 15.047699213027954
(349, 1.9364550113677979, 0.0016400950915674319)
(349, 3.93871808052063, 0.001300898887815752)
(349, 5.860158443450928, 0.0015589822792300802)
(349, 7.784247159957886, 0.0018144510249840096)
(349, 9.711793422698975, 0.0020217249414655474)
(349, 11.62366795539856, 0.0016196743770483502)
(349, 13.521844148635864, 0.0011128773024823631)
epoch time 15.079636335372925
(350, 1.9359312057495117, 0.0018928676866807462)
(350, 3.853344678878784, 0.0014011362253935362)
(350, 5.757847547531128, 0.0011777452051245l7)
```

In [21]:
```
textfile = open("loss2.txt", "w")
for element in losses2:
    textfile.write(str(element) + "\n")
textfile.close()
```

In [14]:

```
import time
import torch.optim as optim
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet32().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses3 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses3.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    print('epoch time',time.time()-s)
print('Finished Training')
```

```
epoch time 16.981651544570923
(348, 2.1738123893737793, 0.005050788302098078)
(348, 4.375619888305664, 0.0020687289483137237)
(348, 6.552916526794434, 0.002176484085497095)
(348, 8.717389583587646, 0.002841057912185218)
(348, 10.906378746032715, 0.004713833947993853)
(348, 13.078289031982422, 0.005461510687493909)
(348, 15.234283208847046, 0.004482235760090644)
epoch time 16.993855476379395
(349, 2.15759015083313, 0.005680434474108589)
(349, 4.330235242843628, 0.006009943694189875)
(349, 6.558507919311523, 0.007948258941000024)
(349, 8.713350534439087, 0.0064665137342300874)
(349, 10.95883822441101, 0.006662871262144146)
(349, 13.132500410079956, 0.00519731833851345)
(349, 15.293569564819336, 0.007836620545499407)
epoch time 17.066230297088623
(350, 2.193568229675293, 0.006381992861267408)
(350, 4.3531880378723145, 0.0053785650747975484)
(350, 6.49826455116272, 0.003700878180988722)
```

In [22]:
```
textfile = open("loss3.txt", "w")
for element in losses3:
    textfile.write(str(element) + "\n")
textfile.close()
```

In [16]:

```python
import time
import torch.optim as optim
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet44().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses4 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses4.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    print('epoch time',time.time()-s)
print('Finished Training')
```

```
epoch time 19.30081868171692
(348, 2.4183807373046875, 0.00026019748452663590)
(348, 4.85472297668457, 0.00026425342429672817)
(348, 7.2866370677948, 0.0002358581078457361)
(348, 9.75323128700256, 0.0002502906684881333)
(348, 12.216313123703003, 0.00026219147143912813)
(348, 14.651654958724976, 0.0002639513215250858)
(348, 17.15526032447815, 0.0003367380144185035)
epoch time 19.15051579475403
(349, 2.4355103969573975, 0.0003476006981034107 4)
(349, 4.880463361740112, 0.0001450401574699327)
(349, 7.329374074935913, 0.0003974822815953117 4)
(349, 9.773640394210815, 0.0002421340493815570 8)
(349, 12.22969102859497, 0.0002420310777278227 4)
(349, 14.669119358062744, 0.0001946302176181735 3)
(349, 17.112865924835205, 0.0002193484175890893 3)
epoch time 19.118512868881226
(350, 2.4492437839508057, 0.0002366859135895074 2)
(350, 4.93298602104187, 0.0001306754963238467)
(350, 7.421787261962891, 0.0001578426717412871 9)
```

In [23]:
```
textfile = open("loss4.txt", "w")
for element in losses4:
    textfile.write(str(element) + "\n")
textfile.close()
```

In [18]:

```
import time
import torch.optim as optim
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet56().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses5 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses5.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    print('epoch time',time.time()-s)
print('Finished Training')
```

```
epoch time 21.14885187149048
(348, 2.7219815254211426, 0.00013521026914754564)
(348, 5.428853273391724, 0.000286350500517779)
(348, 8.151169300079346, 0.00011966762743826139)
(348, 10.885328531265259, 0.0001462166046436249)
(348, 13.623244762420654, 0.00028213543003273244)
(348, 16.37775468826294, 0.00025655352770367505)
(348, 19.106053590774536, 0.00015989755626592063)
epoch time 21.332329750061035
(349, 2.740628242492676, 0.0003432557903542314)
(349, 5.478779077529907, 0.00016537508560889234)
(349, 8.230273723602295, 0.00020195815500765278)
(349, 10.978095293045044, 0.00011975838684260712)
(349, 14.93930721282959, 0.00014468777816527647)
(349, 17.750129461288452, 0.00014860440536054108)
(349, 20.486787796020508, 0.00027546815077325197)
epoch time 22.712486028671265
(350, 2.745670795440674, 0.0005892272207087704)
(350, 5.470365762710571, 0.0001469808838158556)
(350, 8.200093030929565, 0.00017806256867942 82)
```

In [1]:
```
textfile = open("loss5.txt", "w")
for element in losses5:
    textfile.write(str(element) + "\n")
textfile.close()
```

...

In [5]:

```python
import time
import torch.optim as optim
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet50().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses6 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses6.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    print('epoch time',time.time()-s)
print('Finished Training')
```

```
(347, 19.128620147705078, 0.00036499017124344461)
epoch time 21.215643405914307
(348, 2.61718487739563, 0.0006877457737424934)
(348, 5.144724607467651, 0.0002900304242182162)
(348, 7.704703330993652, 0.0007437641512037122)
(348, 10.253649473190308, 0.0004070300750086341)
(348, 12.803223609924316, 0.00045129853554369347)
(348, 15.361856460571289, 0.00034098081484945414)
(348, 17.948991537094116, 0.0007296348025823934)
epoch time 20.02879571914673
(349, 2.534803628921509, 0.0004356831294509383)
(349, 5.063356399536133, 0.0002361861720137127)
(349, 7.575986385345459, 0.00043401876179387374)
(349, 10.115673542022705, 0.00028960825132956844)
(349, 12.656275987625122, 0.0003594618736278047)
(349, 15.185049295425415, 0.0004238990606646272)
(349, 17.738530158996582, 0.00026988007314740774)
epoch time 19.840256690979004
(350, 2.552741289138794, 0.0006491960738625197)
(350, 5.088864088058472, 0.0012246367715510577)
```

In [6]:

```
textfile = open("loss6.txt", "w")
for element in losses6:
    textfile.write(str(element) + "\n")
textfile.close()
```

In [ ]:

```

```

In [17]:

```
import time
import torch.optim as optim
import torchvision.models as models

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = models.resnet50().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses6 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses6.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
    train_acc = 0
    count = 0.0
    for i, data in enumerate(testloader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, y_pred = torch.max(outputs, 1)
#         print(y_pred.shape,labels.shape)
        train_acc += torch.sum(y_pred == labels)
        count+=len(y_pred)
    train_acc=float(train_acc/count)
    print('Acc',train_acc)
    if(train_acc>0.92):
        break
    print('epoch time',time.time()-s,train_acc/count)
print('Finished Training')
```

```
cuda:0
(1, 3.0936830043792725, 3.122834215358812)
(1, 6.160975694656372, 2.3637631620679582)
(1, 9.212742805480957, 2.241091256238976)
```

```
(1, 12.284221649169922, 2.1585688882944534)
(1, 15.339528799057007, 2.1069597614054776)
(1, 18.41964817047119, 2.079439844403948)
(1, 21.48569631576538, 1.9995545951687559)
Acc 0.31610000133514404
epoch time 26.452991008758545 3.1610000133514406e-05
(2, 3.066366672515869, 1.926871650072993)
(2, 6.19232702255249, 1.895579634880533)
(2, 9.281157732009888, 1.895548282837381)
(2, 12.345937490463257, 1.8633296635686134)
(2, 15.401228904724121, 1.8469602404808512)
(2, 18.446027517318726, 1.792870536142466)
(2, 21.51021122932434, 1.8003771597025346)
Acc 0.3610999882221222
epoch time 26.551684856414795 3.6109998822212222e-05
```

In [5]:

```
import time
import torch.optim as optim
import torchvision.models as models

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = resnet50().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
losses6 = []
for epoch in range(350):  # loop over the dataset multiple times
    s = time.time()
    running_loss = 0.0
    train_acc = 0
    count = 0.0
    for i, data in enumerate(trainloader, 0):
#         print(i)
              # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses6.append(loss.item())
        if i % 50 == 49:    # print every 2000 mini-batches
            print((epoch + 1, time.time()-s, running_loss/49))
            running_loss = 0.0
#     train_acc = 0
#     count = 0.0
#     for i, data in enumerate(testloader, 0):
#         inputs, labels = data
#         inputs, labels = inputs.to(device), labels.to(device)
#         outputs = model(inputs)
        _, y_pred = torch.max(outputs, 1)
#         print(y_pred.shape,labels.shape)
        train_acc += torch.sum(y_pred == labels)
        count+=len(y_pred)
    train_acc=float(train_acc/count)
    print('Acc',train_acc)
    if(train_acc>0.92):
        break
    print('epoch time',time.time()-s,train_acc/count)
print('Finished Training')
```

```
(33, 7.665208578109741, 0.2638191377021828 5)
(33, 10.22897505760 1929, 0.28887347390457074)
(33, 12.67027759552002, 0.28931142268132193)
(33, 15.210508346557617, 0.29571513375457453)
```

```
(33, 17.721519470214844, 0.29120015428990736)
Acc 0.9042800068855286
epoch time 19.78704047203064 1.808560013771057e-05
(34, 2.5666356086730957, 0.2451854366428998)
(34, 5.6526148319244385, 0.2415621508749164)
(34, 8.839743614196777, 0.23631427695556562)
(34, 11.481876373291016, 0.2559019613022707)
(34, 14.03534746170044, 0.286735474151008)
(34, 16.591984748840332, 0.27381640094883586)
(34, 19.085593223571777, 0.27188572956591234)
Acc 0.9103999733924866
epoch time 21.150903463363647 1.820799946784973e-05
(35, 2.5165305137634277, 0.22343446770492864)

(35, 4.952503204345703, 0.21413426222849866)
(35, 7.449020147323608, 0.2245035360054094)
```

In [ ]:

```

```

In [53]:

```
losses1 = []
textfile = open("loss1.txt", "r")
for element in textfile:
    losses1.append(float(element))
# print(losses1)
textfile.close()
```

In [54]:

```
losses2 = []
textfile = open("loss2.txt", "r")
for element in textfile:
    losses2.append(float(element))
# print(losses2)
textfile.close()
```

In [55]:

```
losses3 = []
textfile = open("loss3.txt", "r")
for element in textfile:
    losses3.append(float(element))
# print(losses3)
textfile.close()
```

In [56]:

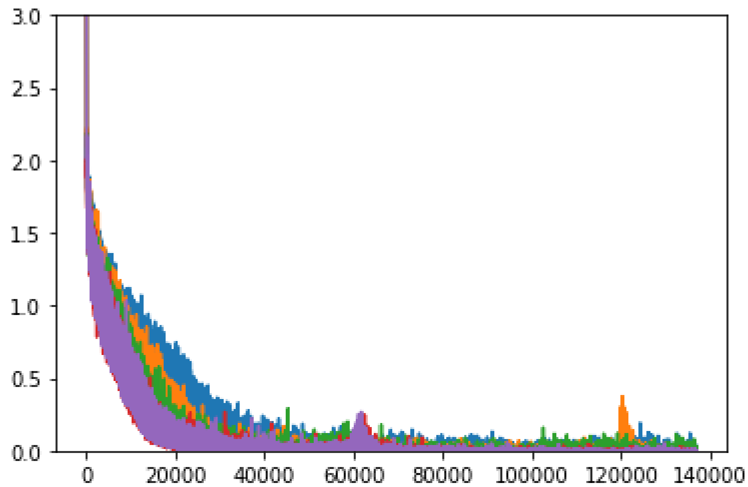
```
losses4 = []
textfile = open("loss4.txt", "r")
for element in textfile:
    losses4.append(float(element))
# print(losses4)
textfile.close()
```

In [57]:

```
losses5 = []
textfile = open("loss5.txt", "r")
for element in textfile:
    losses5.append(float(element))
# print(losses5)
textfile.close()
```

In [58]:
```
import matplotlib.pyplot as plt

plt.plot(range(len(losses1)),losses1)
plt.plot(range(len(losses2)),losses2)
plt.plot(range(len(losses3)),losses3)
plt.plot(range(len(losses4)),losses4)
plt.plot(range(len(losses5)),losses5)
plt.ylim(0,3)
```

Out[58]: (0.0, 3.0)

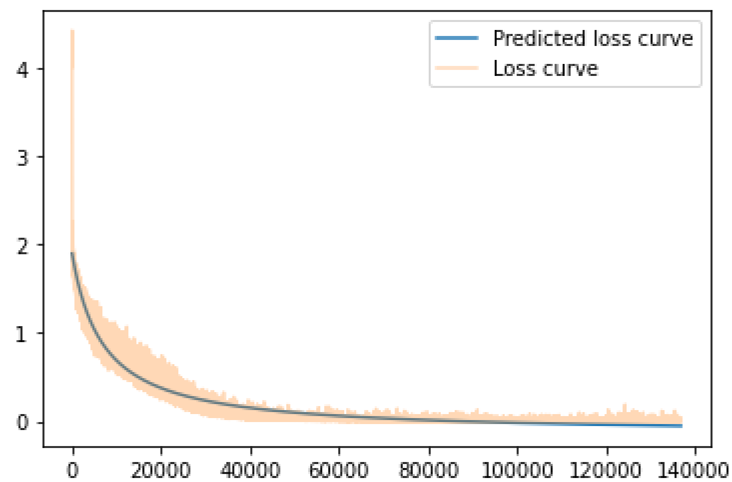


In [59]:
```
v100 = []
```

In [60]:
```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses1)+1),losses1,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
v100.append(popt)
print(popt)
plt.plot(range(1,len(losses1)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses1)+1),losses1,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[ 7.01994062e-05  4.87482273e-01 -1.50206444e-01]
```

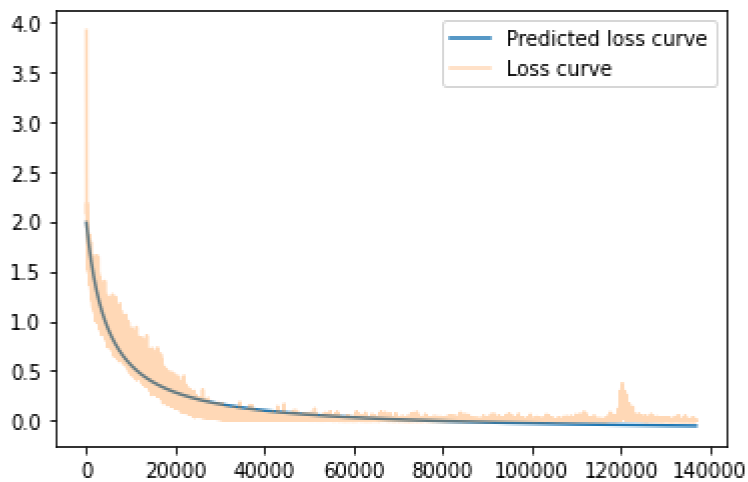
Out[60]: <matplotlib.legend.Legend at 0x7f9718a00b10>



In [61]:
```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses2)+1),losses2,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
v100.append(popt)
print(popt)
plt.plot(range(1,len(losses2)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses2)+1),losses2,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[ 9.96594588e-05  4.74812073e-01 -1.20806639e-01]
```

Out[61]: <matplotlib.legend.Legend at 0x7f9718983e90>



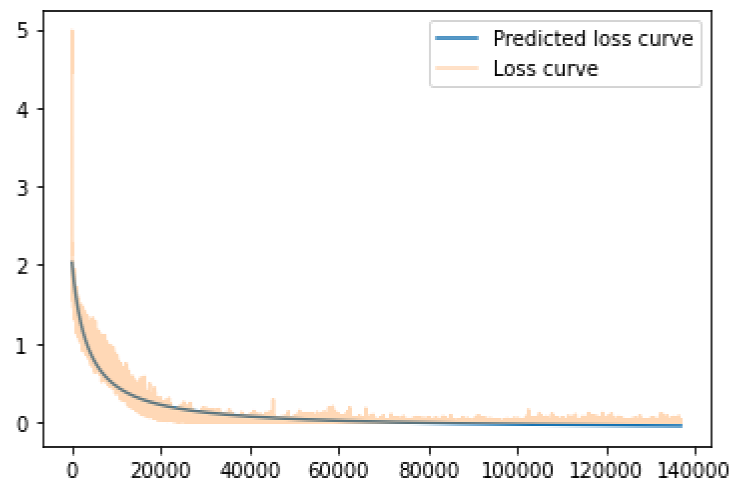
In [62]:

```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses3)+1),losses3,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
v100.append(popt)
print(popt)
plt.plot(range(1,len(losses3)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses3)+1),losses3,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[ 1.32378100e-04  4.69869572e-01 -9.88729486e-02]
```

Out[62]:

```
<matplotlib.legend.Legend at 0x7f97188fe9d0>
```

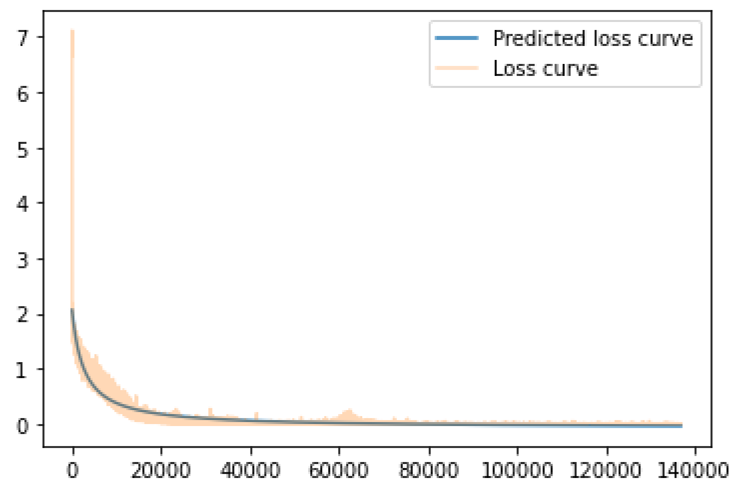


In [63]:

```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses4)+1),losses4,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
v100.append(popt)
print(popt)
plt.plot(range(1,len(losses4)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses4)+1),losses4,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[ 1.72993949e-04  4.67240070e-01 -8.18453202e-02]
```

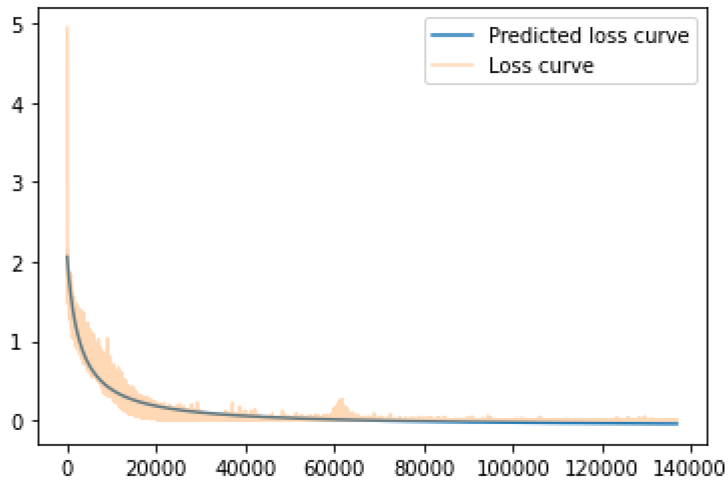
Out[63]: <matplotlib.legend.Legend at 0x7f97189d6c90>



In [64]:
```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses5)+1),losses5,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
v100.append(popt)
print(popt)
plt.plot(range(1,len(losses5)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses5)+1),losses5,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[ 1.63617982e-04  4.65227282e-01 -8.58001223e-02]
```

Out[64]: <matplotlib.legend.Legend at 0x7f971a39a550>



In [65]:
```
print(v100)
```

```
[array([ 7.01994062e-05,  4.87482273e-01, -1.50206444e-01]), array([ 9.96594588
e-05,  4.74812073e-01, -1.20806639e-01]), array([ 1.32378100e-04,  4.69869572e-
01, -9.88729486e-02]), array([ 1.72993949e-04,  4.67240070e-01, -8.18453202e-0
2]), array([ 1.63617982e-04,  4.65227282e-01, -8.58001223e-02])]
```

In [65]:
```

```

In [66]:
```
import pandas as pd

loss_data = pd.read_pickle('merged_result_P100.pickle')
```

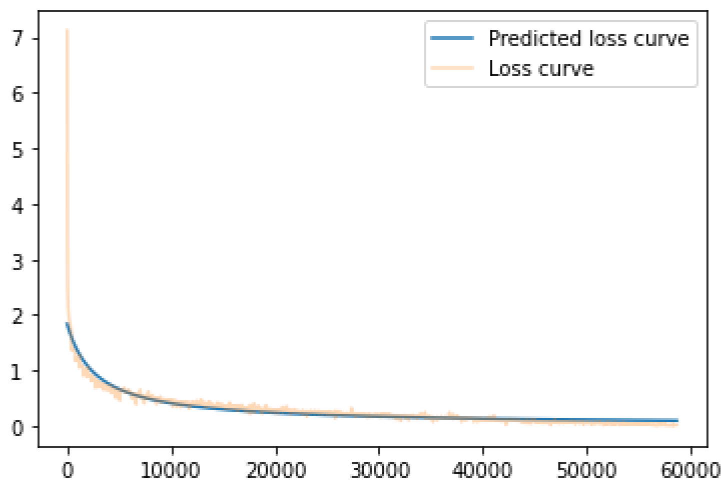
In [67]:
```
loss = []
for i in loss_data:
    if i!='resnet50':
        loss.append(loss_data[i])
loss.append(loss_data['resnet50'])

```

In [68]:
```
p100 = []
losses1,losses2,losses3,losses4,losses5,_= loss
```

In [69]:
```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses1)+1),losses1,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
p100.append(popt)
print(popt)
plt.plot(range(1,len(losses1)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses1)+1),losses1,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[1.98766092e-04 5.48874855e-01 1.76083898e-02]
```

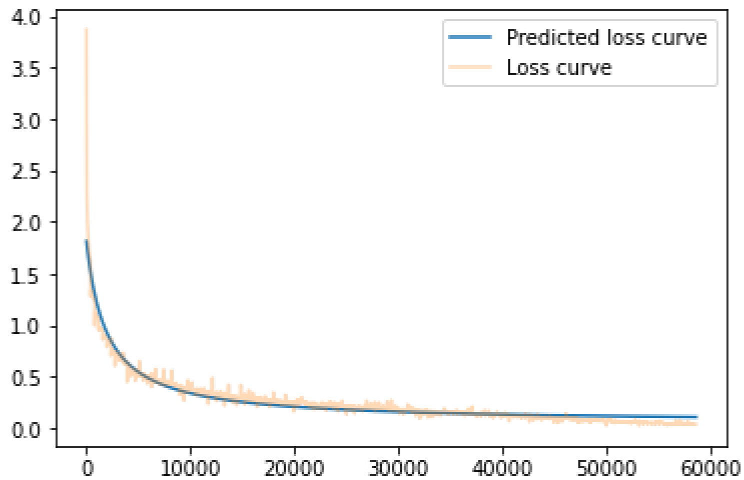
Out[69]: <matplotlib.legend.Legend at 0x7f971df15210>



In [70]:
```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses2)+1),losses2,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
p100.append(popt)
print(popt)
plt.plot(range(1,len(losses2)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses2)+1),losses2,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[2.90696922e-04 5.69169631e-01 5.44017628e-02]
```

Out[70]: <matplotlib.legend.Legend at 0x7f971dd69e10>

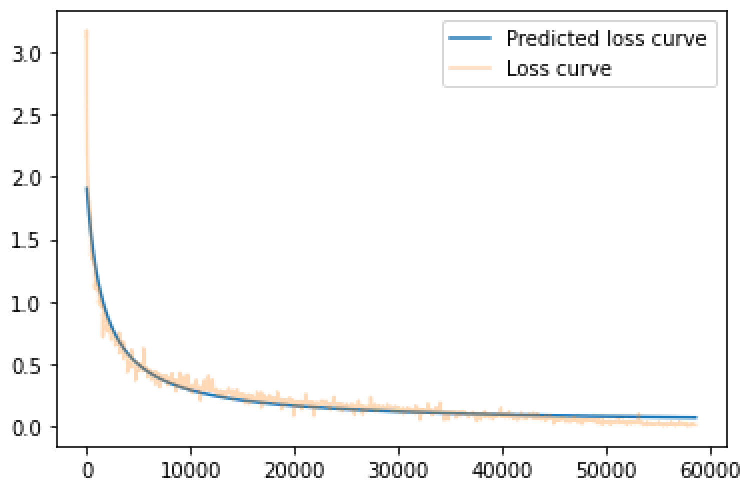


In [71]:

```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses3)+1),losses3,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
p100.append(popt)
print(popt)
plt.plot(range(1,len(losses3)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses3)+1),losses3,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[3.15287961e-04 5.30339862e-01 2.07627717e-02]
```

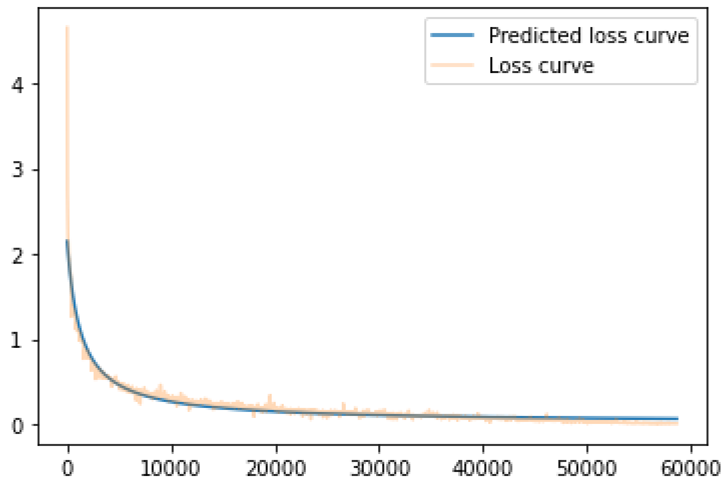
Out[71]: <matplotlib.legend.Legend at 0x7f9718351dd0>



In [72]:
```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses4)+1),losses4,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
p100.append(popt)
print(popt)
plt.plot(range(1,len(losses4)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses4)+1),losses4,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[3.57152914e-04 4.70425416e-01 1.78155181e-02]
```

Out[72]: <matplotlib.legend.Legend at 0x7f9722216950>

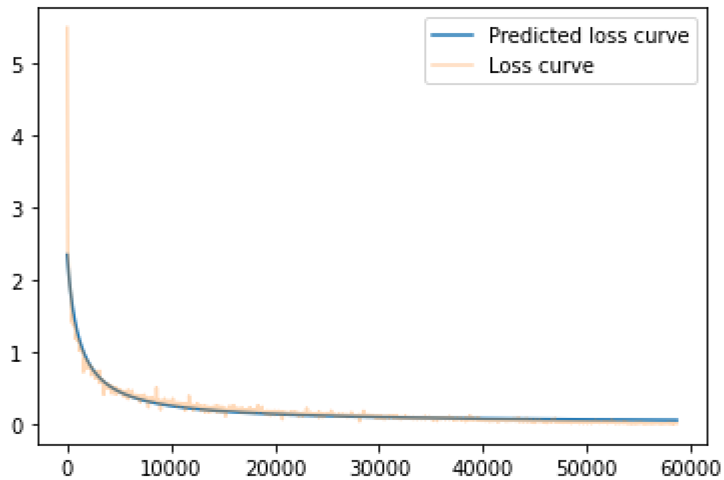


In [73]:

```
from scipy.optimize import curve_fit
def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses5)+1),losses5,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
p100.append(popt)
print(popt)
plt.plot(range(1,len(losses5)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses5)+1),losses5,alpha=0.3,label='Loss curve')
plt.legend()
```

```
[3.68076907e-04 4.27519868e-01 8.02674203e-03]
```

Out[73]: <matplotlib.legend.Legend at 0x7f9719726b90>



In [74]:

```
print(v100)
print(p100)
```

```
[array([ 7.01994062e-05,  4.87482273e-01, -1.50206444e-01]), array([ 9.96594588
e-05,  4.74812073e-01, -1.20806639e-01]), array([ 1.32378100e-04,  4.69869572e-
01, -9.88729486e-02]), array([ 1.72993949e-04,  4.67240070e-01, -8.18453202e-0
2]), array([ 1.63617982e-04,  4.65227282e-01, -8.58001223e-02])]
[array([1.98766092e-04, 5.48874855e-01, 1.76083898e-02]), array([2.90696922e-0
4, 5.69169631e-01, 5.44017628e-02]), array([3.15287961e-04, 5.30339862e-01, 2.0
7627717e-02]), array([3.57152914e-04, 4.70425416e-01, 1.78155181e-02]), array
([3.68076907e-04, 4.27519868e-01, 8.02674203e-03])]
```

In [74]:

```

```

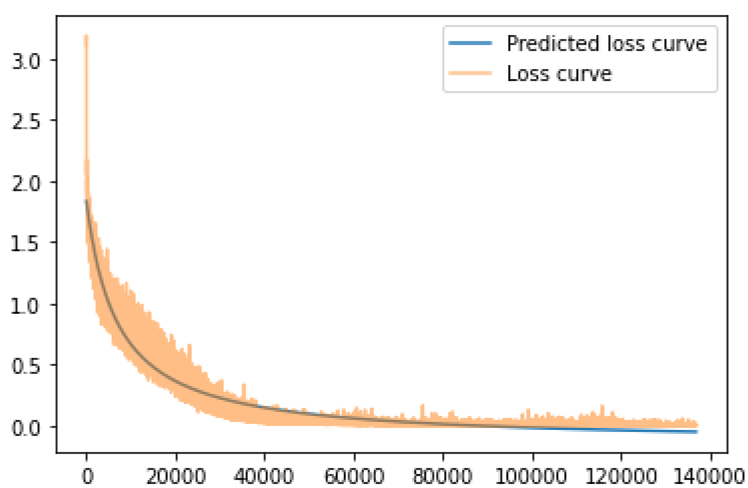
In [75]:
```
# k80
import numpy as np

losses1 = np.load('resnet18_losses.npz')
losses2 = np.load('resnet20_losses.npz')
losses3 = np.load('resnet32_losses.npz')
losses4 = np.load('resnet44_losses.npz')
losses5 = np.load('resnet56_losses.npz')
losses1 = losses1.f.arr_0
losses2 = losses2.f.arr_0
losses3 = losses3.f.arr_0
losses4 = losses4.f.arr_0
losses5 = losses5.f.arr_0
# print(losses1.keys)
k80 = []

losses6 = np.load('resnet50_new.npz')
losses6 = losses6.f.a
print(losses6)
```

```
[7.00012493e+00 6.51588106e+00 6.20615292e+00 ... 4.14951630e-02
 3.50350663e-02 4.87744343e-03]
```

In [76]:
```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses1)+1),losses1,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
k80.append(popt)
print(popt)
plt.plot(range(1,len(losses1)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses1)+1),losses1,alpha=0.5,label='Loss curve')
plt.legend()
```

```
[ 7.25235662e-05  5.05014458e-01 -1.48671083e-01]
```

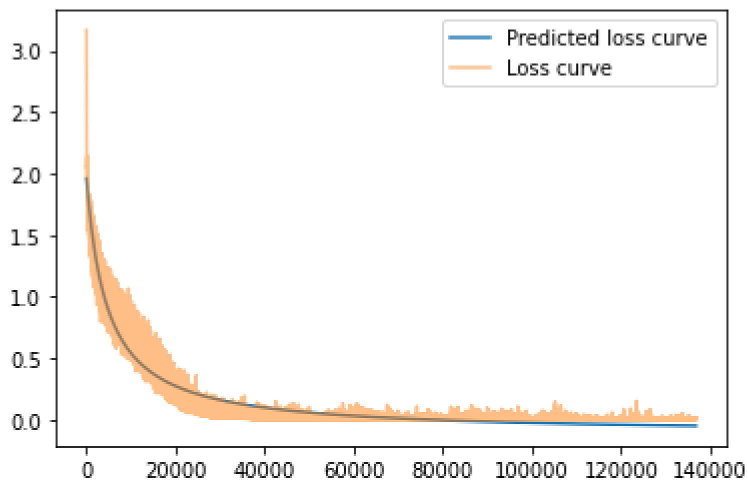
Out[76]: <matplotlib.legend.Legend at 0x7f971e77c550>



In [77]:
```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses2)+1),losses2,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
k80.append(popt)
print(popt)
plt.plot(range(1,len(losses2)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses2)+1),losses2,alpha=0.5,label='Loss curve')
plt.legend()
```

```
[ 1.03425344e-04  4.81340265e-01 -1.18030695e-01]
```

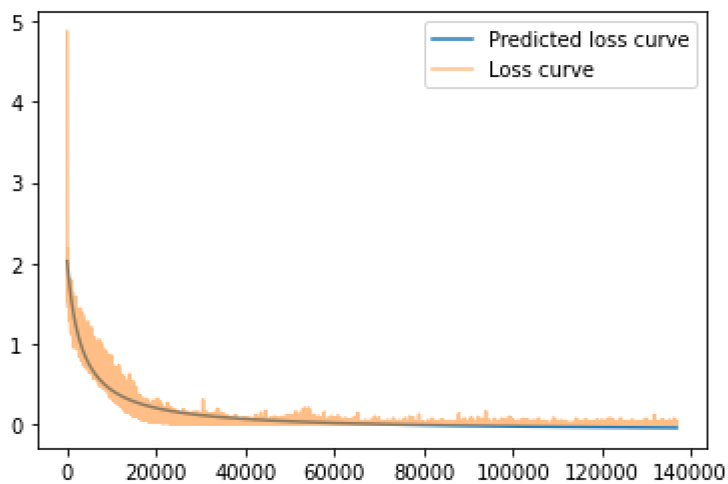
Out[77]: <matplotlib.legend.Legend at 0x7f9722223e90>



In [78]:
```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses3)+1),losses3,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
k80.append(popt)
print(popt)
plt.plot(range(1,len(losses3)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses3)+1),losses3,alpha=0.5,label='Loss curve')
plt.legend()
```

```
[ 1.46424549e-04  4.71928719e-01 -9.19557722e-02]
```

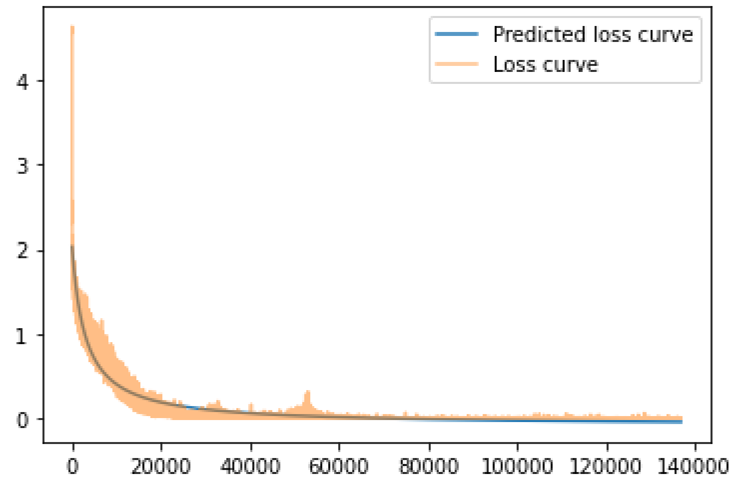
Out[78]: <matplotlib.legend.Legend at 0x7f971e5f5e50>



In [79]:
```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses4)+1),losses4,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
k80.append(popt)
print(popt)
plt.plot(range(1,len(losses4)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses4)+1),losses4,alpha=0.5,label='Loss curve')
plt.legend()
```

```
[ 1.56356261e-04  4.71935887e-01 -8.90760333e-02]
```

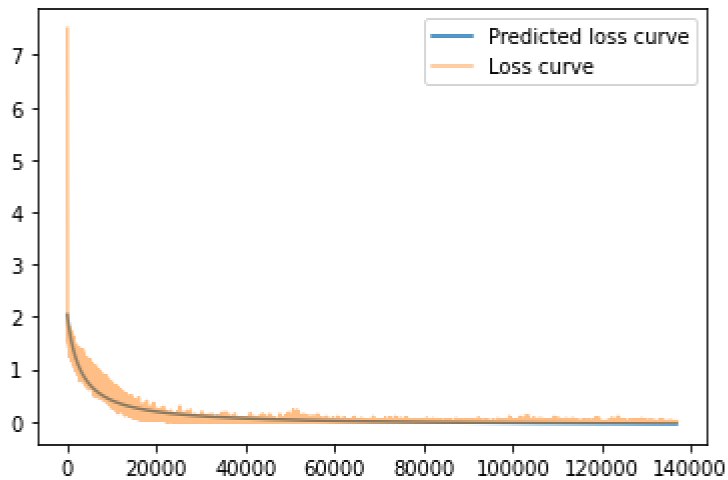
Out[79]: <matplotlib.legend.Legend at 0x7f971e5717d0>



In [80]:
```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def func(k,beta0,beta1,beta2):
    return (1/(beta0*k+beta1)) + beta2
popt, pcov = curve_fit(func, range(1,len(losses5)+1),losses5,p0=[0.001, 0.1,
beta0,beta1,beta2 = popt
k80.append(popt)
print(popt)
plt.plot(range(1,len(losses5)+1),[func(k,beta0,beta1,beta2) for k in range(1
plt.plot(range(1,len(losses5)+1),losses5,alpha=0.5,label='Loss curve')
plt.legend()
```

```
[ 1.55769825e-04  4.69468259e-01 -8.89310787e-02]
```

Out[80]: <matplotlib.legend.Legend at 0x7f971e695950>

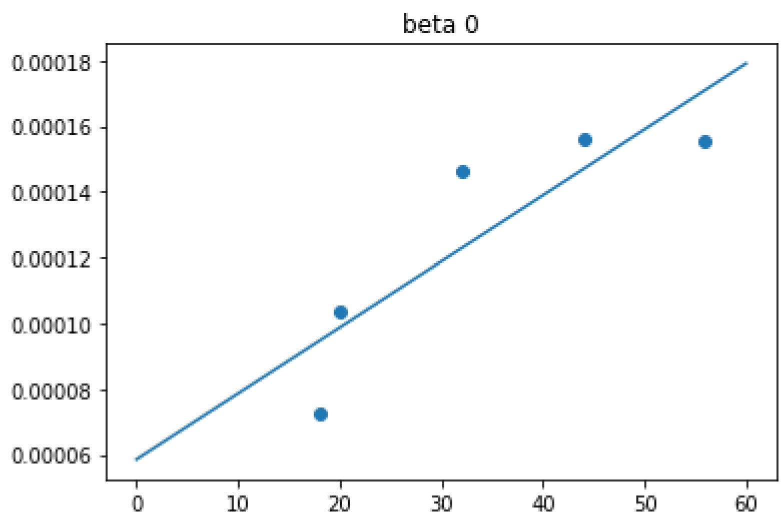


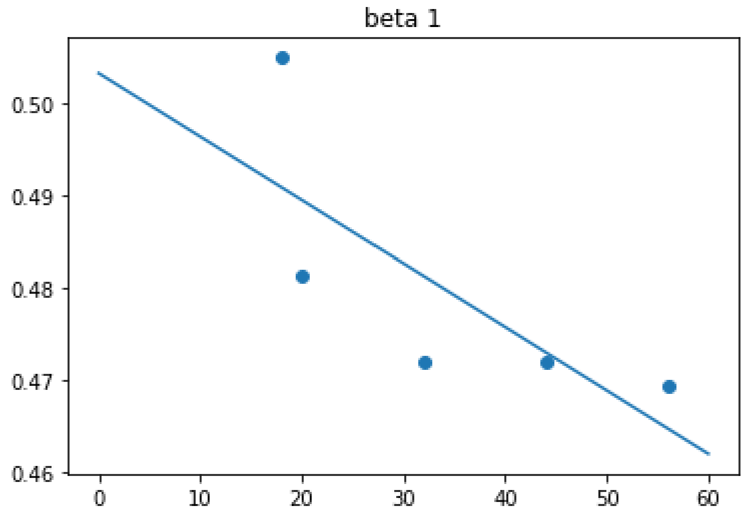
In [81]:
```
l_k80 = np.array([[18],[20],[32],[44],[56]])
l_p100 = np.array([[18],[20],[32],[44],[56]])
l_v100 = np.array([[18],[20],[32],[44],[56]])
```

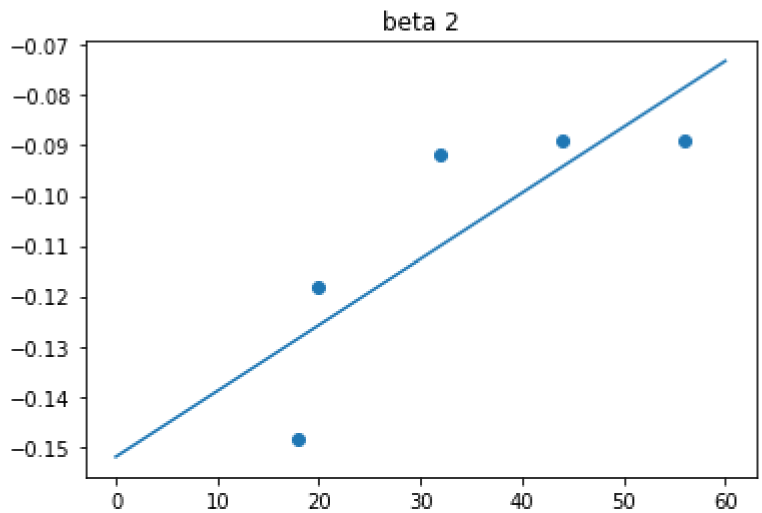
In [82]:
```
b0_k80 = [i[0] for i in k80]
b1_k80 = [i[1] for i in k80]
b2_k80 = [i[2] for i in k80]

b0_p100 = [i[0] for i in p100]
b1_p100 = [i[1] for i in p100]
b2_p100 = [i[2] for i in p100]

b0_v100 = [i[0] for i in v100]
b1_v100 = [i[1] for i in v100]
b2_v100 = [i[2] for i in v100]

```

In [83]:

```
# k80
from sklearn.linear_model import LinearRegression as lr

x = [[i] for i in range(61)]

kb0 = lr().fit(l_k80,b0_k80)
y = kb0.predict(x)
plt.plot(x,y)
plt.scatter(l_k80,b0_k80)
plt.title('beta 0')
plt.show()

kb1 = lr().fit(l_k80,b1_k80)
y = kb1.predict(x)
plt.plot(x,y)
plt.scatter(l_k80,b1_k80)
plt.title('beta 1')
plt.show()

kb2 = lr().fit(l_k80,b2_k80)
y = kb2.predict(x)
plt.plot(x,y)
plt.scatter(l_k80,b2_k80)
plt.title('beta 2')
plt.show()
```

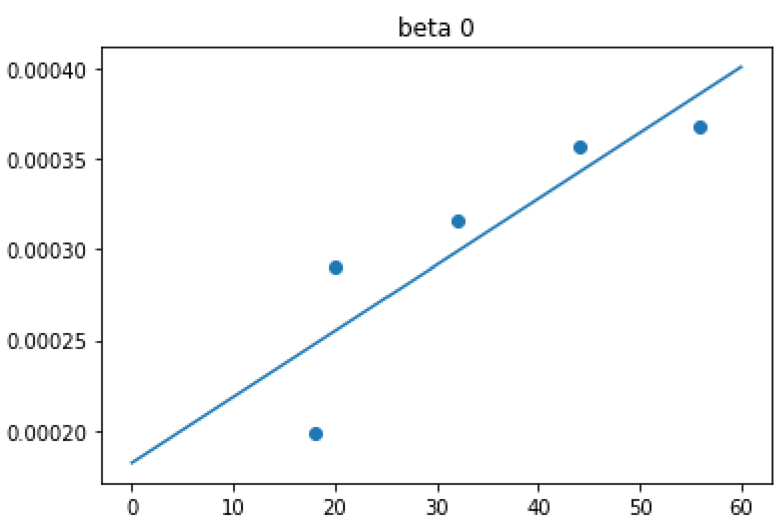


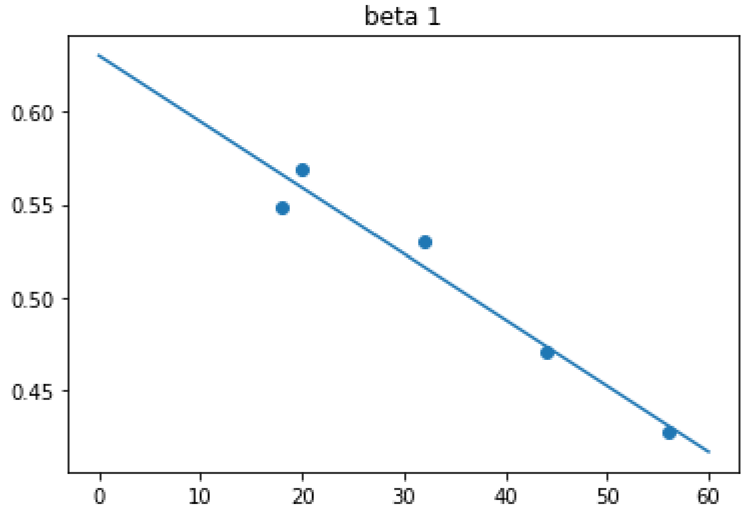

beta 1
0.50
0.49
0.48
0.47
0.46
0
10
20
30
40
50
60

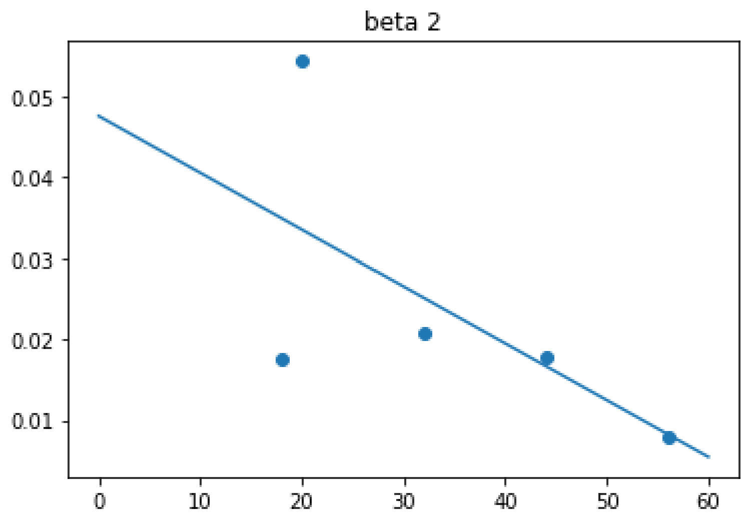

beta 2
−0.07
−0.08
−0.09
−0.10
−0.11
−0.12
−0.13
−0.14
−0.15
0
10
20
30
40
50
60

In [84]:

```
# p100
from sklearn.linear_model import LinearRegression as lr

x = [[i] for i in range(61)]

pb0 = lr().fit(l_p100,b0_p100)
y = pb0.predict(x)
plt.plot(x,y)
plt.scatter(l_p100,b0_p100)
plt.title('beta 0')
plt.show()

pb1 = lr().fit(l_p100,b1_p100)
y = pb1.predict(x)
plt.plot(x,y)
plt.scatter(l_p100,b1_p100)
plt.title('beta 1')
plt.show()

pb2 = lr().fit(l_p100,b2_p100)
y = pb2.predict(x)
plt.plot(x,y)
plt.scatter(l_p100,b2_p100)
plt.title('beta 2')
plt.show()
```

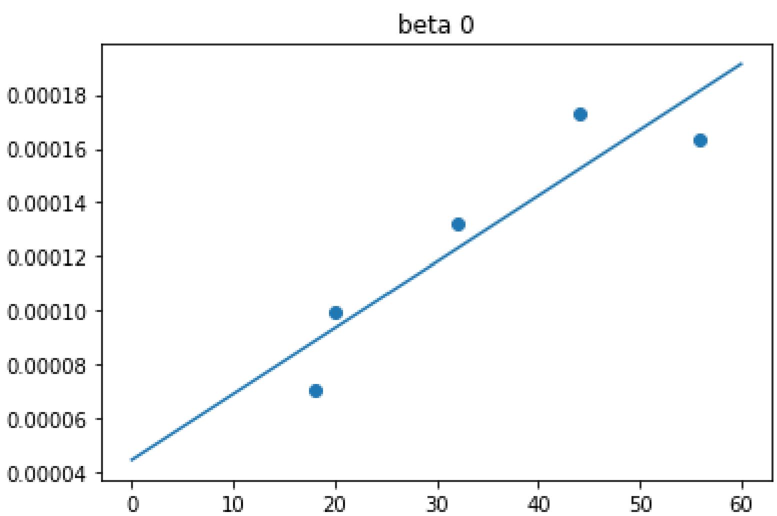


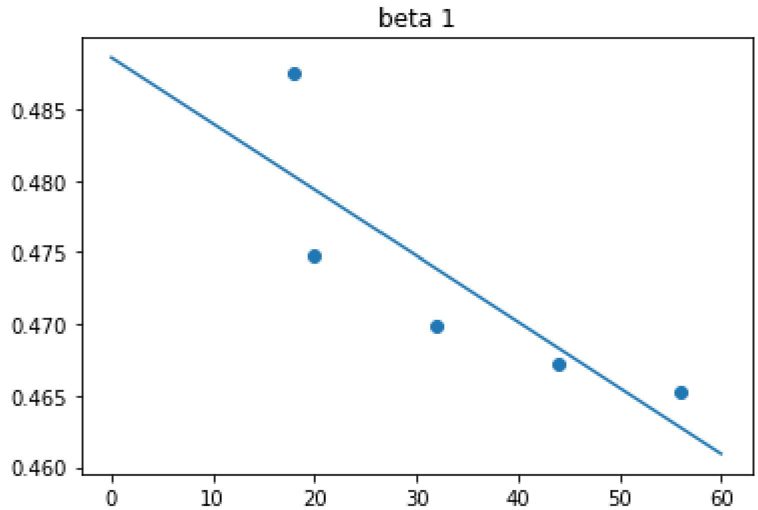

beta 1
0.60
0.55
0.50
0.45
0
10
20
30
40
50
60

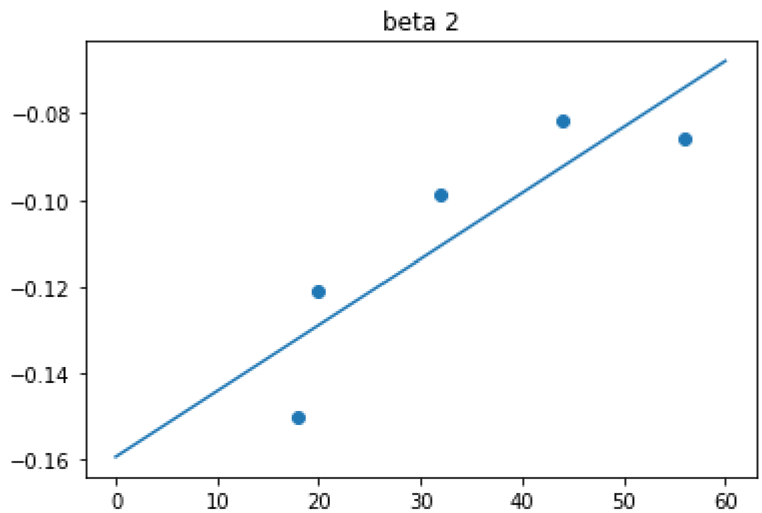

beta 2
0.05
0.04
0.03
0.02
0.01
0
10
20
30
40
50
60

In [85]:

```
# v100
from sklearn.linear_model import LinearRegression as lr

x = [[i] for i in range(61)]


vb0 = lr().fit(l_v100,b0_v100)
y = vb0.predict(x)
plt.plot(x,y)
plt.scatter(l_v100,b0_v100)
plt.title('beta 0')
plt.show()

vb1 = lr().fit(l_v100,b1_v100)
y = vb1.predict(x)
plt.plot(x,y)
plt.scatter(l_v100,b1_v100)
plt.title('beta 1')
plt.show()

vb2 = lr().fit(l_v100,b2_v100)
y = vb2.predict(x)
plt.plot(x,y)
plt.scatter(l_v100,b2_v100)
plt.title('beta 2')
plt.show()
```







In [86]:

```
# Predictions for ResNet50
k80beta = [kb0.predict([[50]]),kb1.predict([[50]]),kb2.predict([[50]])]
p100beta = [pb0.predict([[50]]),pb1.predict([[50]]),pb2.predict([[50]])]
v100beta = [vb0.predict([[50]]),vb1.predict([[50]]),vb2.predict([[50]])]
print(k80beta)
print(p100beta)
print(v100beta)
```

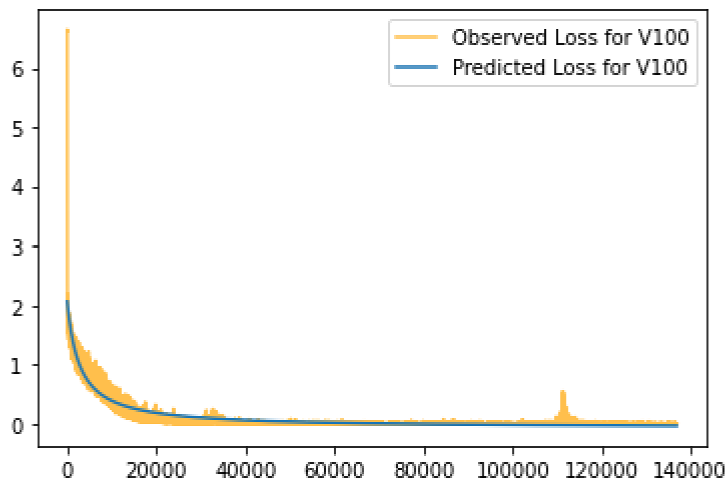
```
[array([0.00015904]), array([0.46893456]), array([-0.08628932])]
[array([0.00036428]), array([0.45232191]), array([0.0124901])]
[array([0.00016694]), array([0.46555051]), array([-0.08310189])]
```

In [87]:

```
losses6 = []
textfile = open("loss6.txt", "r")
for element in textfile:
    losses6.append(float(element))
textfile.close()
```

In [88]:
```
# v100

l = []
err = []
epochs = len(losses6)
for i in range(epochs):
    temp = func(i+1,v100beta[0],v100beta[1],v100beta[2])
    l.append(temp)
    err.append((temp-losses6[i])/losses6[i])
plt.plot([i+1 for i in range(epochs)],losses6,alpha=0.7,color='orange',label
plt.plot([i+1 for i in range(epochs)],l,label='Predicted Loss for V100')
plt.legend()
# Percentage error

```

Out[88]: <matplotlib.legend.Legend at 0x7f9718edc6d0>



In [97]:
```
def finv(l,b0,b1,b2):
    return ((1/(l-b2))-b1)/b0
# For V100 loss at 92% accuracy = 0.27
print(finv(0.27,v100beta[0],v100beta[1],v100beta[2]),' iterations')
print('Which is equal to ',finv(0.27,v100beta[0],v100beta[1],v100beta[2])//3

```

```
[14175.33011273]  iterantions
Which is equal to  [36.]  epochs
```

As calculated the model reaches 92% accuracy after epoch 36. Percentage error = 0%

In [88]:
```

```

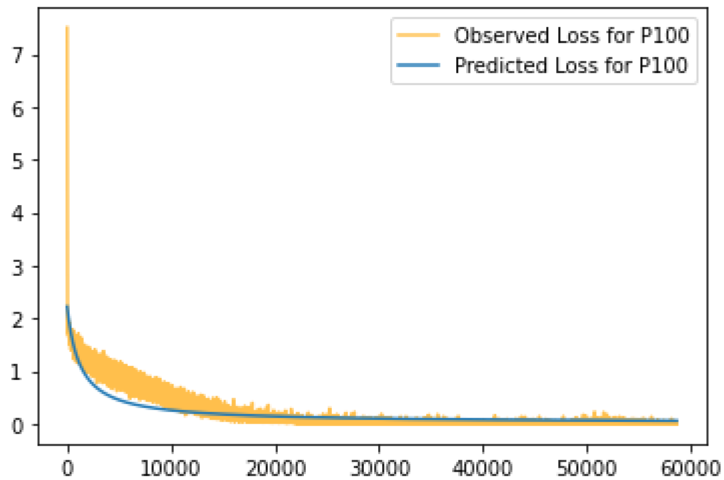
In [92]:
```
losses6 = loss_data['resnet50']
```

In [95]:

```
# p100

l = []
err = []
epochs = len(losses6)
for i in range(epochs):
    temp = func((i+1),p100beta[0],p100beta[1],p100beta[2])
    l.append(temp)
    err.append((temp-losses6[i])/losses6[i])
plt.plot([i+1 for i in range(epochs)],losses6,alpha=0.7,color='orange',label
plt.plot([i+1 for i in range(epochs)],l,label='Predicted Loss for P100')
plt.legend()
# Percentage error
```

Out[95]:

```
<matplotlib.legend.Legend at 0x7f971dff9450>
```



In [120]:

```
def finv(l,b0,b1,b2):
    return ((1/(l-b2))-b1)/b0
# For P100 loss at 92% accuracy = 0.18
print(finv(0.18,p100beta[0],p100beta[1],p100beta[2]),' iterations')
print('Which is equal to ',finv(0.18,p100beta[0],p100beta[1],p100beta[2])//3

```

```
[15146.13008173]  iterations
Which is equal to  [38.]  epochs
```

In [ ]:

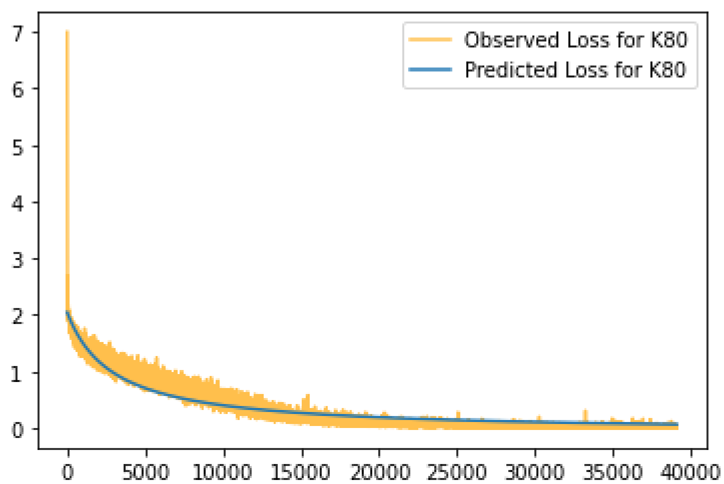
```

```

In [90]:

```
losses6 = np.load('resnet50_new.npz')
losses6 = losses6.f.a
```

In [91]:
```
# k80

l = []
err = []
epochs = len(losses6)
for i in range(epochs):
    temp = func((i+1),k80beta[0],k80beta[1],k80beta[2])
    l.append(temp)
    err.append((temp-losses6[i])/losses6[i])
plt.plot([i+1 for i in range(epochs)],losses6[:epochs],alpha=0.7,color='oran
plt.plot([i+1 for i in range(epochs)],l,label='Predicted Loss for K80')
plt.legend()
# Percentage error
```

Out[91]: <matplotlib.legend.Legend at 0x7f971ee01450>



In [118]:
```
def finv(l,b0,b1,b2):
    return ((1/(l-b2))-b1)/b0
# For P100 loss at 92% accuracy = 0.23
print(finv(0.23,k80beta[0],k80beta[1],k80beta[2]),' iterations')
print('Which is equal to ',finv(0.23,k80beta[0],k80beta[1],k80beta[2])//391,

```

```
[16930.7875504]  iterations
Which is equal to  [43.]  epochs
```

92% occurs at 36 epochs. error = 19.4%

In [ ]:
```

```

In [121]:
```
def f(p,w):
    return (1.02*(128/w) + 2.78 + 4.92*w/p + 0*w + 0.02*p)**-1
```

In [122]:
```
import matplotlib.pyplot as plt

w = range(1,51)
# 92% occurs at epoch 36
epoch = 36
plt.plot(w,[epoch/f(2,w[i]) for i in range(50)],label='2 Parameter Servers')
plt.plot(w,[epoch/f(4,w[i]) for i in range(50)],label='4 Parameter Servers')
plt.xlabel('no of workers')
plt.ylabel('time')
plt.legend()
```

Out[122]: <matplotlib.legend.Legend at 0x7f971e1e3310>