# Final Project Presentation
# -ResNet-

2020095178 최윤선

Colab Link: https://colab.research.google.com/drive/1osTSRTRdpXRdUTCvDNnAjwDhkjxbx7xv?usp=sharing

Reference Paper: Deep Residual Learning for Image Recognition (ResNet)

# Data Normalization

- For better data normalization, I used the mean and standard deviation of the R, G, B data of the train dataset, instead of 0.5.

```
def normalization(dataset):
  mean = np.array([np.mean(x.numpy(), axis=(1,2)) for x, _ in dataset])
  r_mean = mean[:, 0].mean()
  g_mean = mean[:, 1].mean()
  b_mean = mean[:, 2].mean()

  std = np.array([np.std(x.numpy(), axis=(1,2)) for x, _ in dataset])
  r_std = std[:, 0].mean()
  g_std = std[:, 0].mean()
  b_std = std[:, 0].mean()

  return (r_mean, g_mean, b_mean), (r_std, g_std, b_std)
```

→ The function of calculating the mean and standard deviation

```
transform = transforms.Compose([transforms.ToTensor()])
trainset = datasets.CIFAR10(root ='./ data', train=True, download=True, transform=transform)

mean, std = normalization(trainset)
print('Mean (R, G, B): ', mean)
print('Standard deviation (R, G, B): ',std)
```

```
Files already downloaded and verified
Mean (R, G, B):  (0.49139965, 0.48215845, 0.4465309)
Standard deviation (R, G, B):  (0.20220213, 0.20220213, 0.20220213)
```

→ The mean and standard deviation of R, G, B

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean, std)])
```

# Data Augmentation

- For higher accuracy, I applied various data augmentation method on train dataset.

```
transform_aug = transforms.Compose([transforms.Resize((32,32)), transforms.RandomHorizontalFlip(),
                                    transforms.RandomRotation(10), transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
                                    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
                                    transforms.ToTensor(), transforms.Normalize(mean, std)])

trainset = datasets.CIFAR10(root ='./ data', train=True, download=True, transform=transform_aug)
valset = datasets.CIFAR10(root ='./ data', train=True, download=True, transform=transform)
testset = datasets.CIFAR10(root ='./ data', train=False, download=True, transform=transform)

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

- `transforms.Resize():` Resize the image.
- `transforms.RandomHorizontalFlip():` Invert left and right with the defined probability of p. (The default value of p is 0.5.)
- `transforms.RandomRotation():` Rotate the images randomly at a given angle.
- `transforms.RandomAffine():` Do a random affine transformation like rotating or moving.
- `transforms.ColorJitter():` Arbitrarily change brightness, contrast, saturation, color tone.

# Data Loader

- Divide the dataset into train, validation, and test dataset.

```
np.random.seed(0)
val_ratio = 0.1
train_size = len(trainset)
indices = list(range(train_size))
split_idx = int(np.floor(val_ratio*train_size))
np.random.shuffle(indices)
train_idx, val_idx = indices[split_idx:], indices[:split_idx]

train_sampler = SubsetRandomSampler(train_idx)
val_sampler = SubsetRandomSampler(val_idx)

batch_size = 128

train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, sampler=train_sampler, num_workers=2)
val_loader = torch.utils.data.DataLoader(valset, batch_size=batch_size, sampler=val_sampler, num_workers=2)
test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=2)
```

- Set the batch size to 128 according to the paper.

# Model Architecture – ResNet44

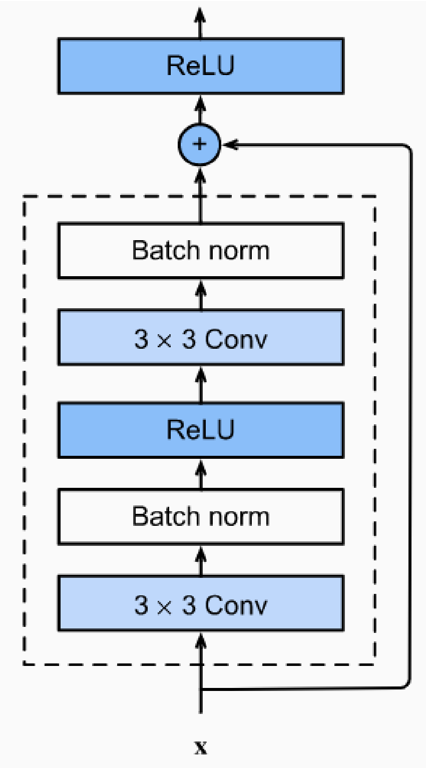
- Implement the commonly used convolutional layer as a function.

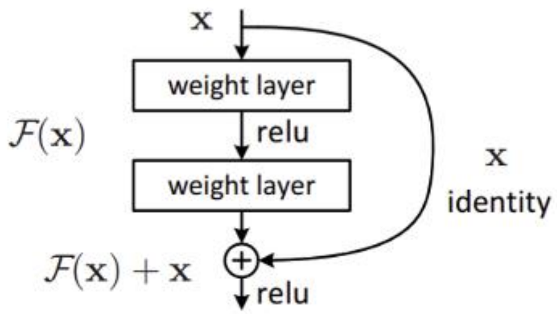
```
def conv3(in_channels, out_channels, stride=1):
  return nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
```

# Model Architecture – ResNet44

- Residual Block

```python
class ResidualBlock(nn.Module):
  def __init__(self, in_channels, out_channels, stride=1, shortcut=None):
    super(ResidualBlock, self).__init__()
    self.conv1 = conv3(in_channels, out_channels, stride)
    self.bn1 = nn.BatchNorm2d(out_channels)
    self.relu = nn.ReLU(inplace=True)
    self.conv2 = conv3(out_channels, out_channels)
    self.bn2 = nn.BatchNorm2d(out_channels)
    self.shortcut = shortcut

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



# Model Architecture – ResNet44

- Main Block

```
class ResNet(nn.Module):
  def __init__(self, block, layers, num_classes=10):
    super(ResNet, self).__init__()
    self.in_channels = 16
    self.conv = conv3(3, self.in_channels)
    self.bn = nn.BatchNorm2d(self.in_channels)
    self.relu = nn.ReLU(inplace=True)
    self.layer1 = self.make_layer(block, 16, layers[0], 1)
    self.layer2 = self.make_layer(block, 32, layers[1], 2)
    self.layer3 = self.make_layer(block, 64, layers[2], 2)
    self.avgPool = nn.AvgPool2d(8)
    self.fc = nn.Linear(64, num_classes)

  def make_layer(self, block, out_channels, num_blocks, stride=1):
    shortcut = None
    if (stride != 1) or (self.in_channels != out_channels):
      shortcut = nn.Sequential(conv3(self.in_channels, out_channels, stride=stride),
                               nn.BatchNorm2d(out_channels))
    layers = []
    layers.append(block(self.in_channels, out_channels, stride, shortcut))
    self.in_channels = out_channels
    for _ in range(1, num_blocks):
      layers.append(block(out_channels, out_channels))
    return nn.Sequential(*layers)

  def forward(self, x):
    out = self.conv(x)
    out = self.bn(out)
    out = self.relu(out)
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.avgPool(out)
    out = out.view(out.size(0), -1)
    out = self.fc(out)
    return out
```

| output map size | $32\times32$ | $16\times16$ | $8\times8$ |
|---|---|---|---|
| # layers | $1+2n$ | $2n$ | $2n$ |
| # filters | 16 | 32 | 64 |

# Model Architecture – ResNet44

- Define the model

```
model = ResNet(ResidualBlock, [7, 7, 7]).to(device)
loss_func = nn.CrossEntropyLoss().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[32000, 48000], gamma=0.1)
```

- According to the paper
  - Set n=7 and create ResNet model with 44 layers.
  - The optimizer is SGD and set the initial learning rate to 0.1, momentum to 0.9, and weight decay to 0.0001.
  - Use `MultiStepLR` method to control the learning rate.

    Decay the learning rate of each parameter group by gamma=0.1,

    once the number of epoch reaches one of the milestones(32000~48000 step).

| output map size | 32×32 | 16×16 | 8×8 |
|---|---|---|---|
| # layers | 1+2$n$ | 2$n$ | 2$n$ |
| # filters | 16 | 32 | 64 |

We use a weight decay of 0.0001 and momentum of 0.9, and adopt the weight initialization in [12] and BN [16] but with no dropout. These models are trained with a mini-batch size of 128 on two GPUs. We start with a learning rate of 0.1, divide it by 10 at 32k and 48k iterations, and

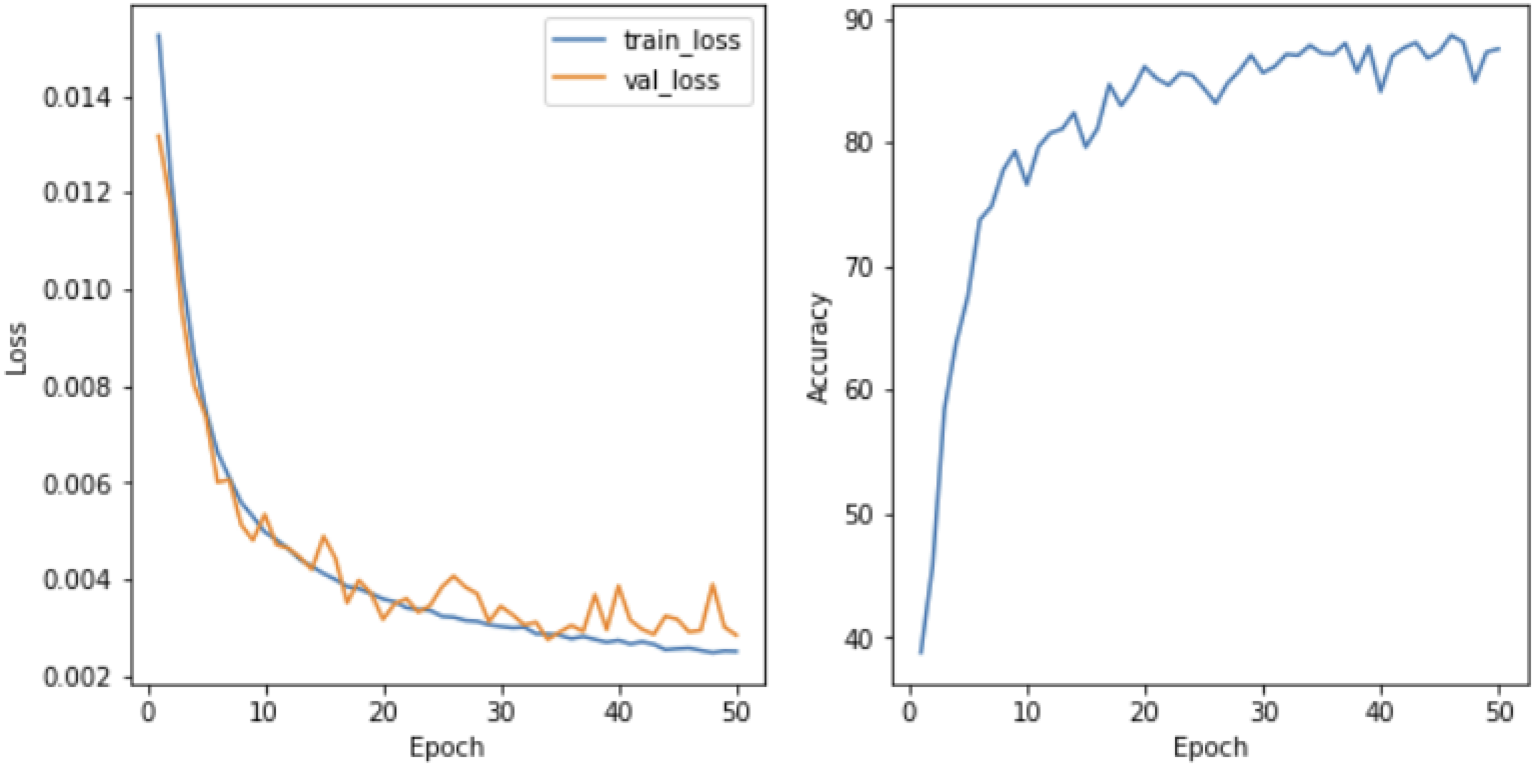
# Train & Validation

```python
epochs = 50
train_loss_list = []
val_loss_list = []
val_acc_list = []

for epoch in range(epochs):
  model.train()
  train_loss = 0
  train_total = 0
  for data, target in train_loader:
    data, target = data.to(device), target.to(device)
    scheduler.step()
    optimizer.zero_grad()
    output = model(data)
    loss = loss_func(output, target)
    loss.backward()
    optimizer.step()
    train_loss += loss.item()
    train_total += target.size(0)

  train_loss /= train_total
  train_loss_list.append(train_loss)

  model.eval()
  correct = 0
  val_total = 0
  val_loss = 0
  with torch.no_grad():
    for data, target in val_loader:
      data, target = data.to(device), target.to(device)
      output = model(data)
      val_loss += loss_func(output, target).item()
      _, prediction = torch.max(output.data, 1)
      val_total += target.size(0)
      correct += prediction.eq(target.view_as(prediction)).sum().item()

  val_loss /= val_total
  val_acc = 100.*correct/val_total

  val_loss_list.append(val_loss)
  val_acc_list.append(val_acc)

  print('Epoch: {}\t Train Loss: {:04f}\t Valid Loss: {:04f}\t Valid Accuracy: {:.2f}'.format(epoch+1, train_loss, val_loss, val_acc))

loss_list = torch.tensor(val_loss_list)
acc_list = torch.tensor(val_acc_list)

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(torch.tensor(range(1,51)), train_loss_list, label='train_loss')
plt.plot(torch.tensor(range(1,51)), val_loss_list, label='val_loss')
plt.legend()
plt.subplot(1,2,2)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.plot(torch.tensor(range(1,51)), val_acc_list)
plt.show()
```

```
Epoch: 1	 Train Loss: 0.015249	 Valid Loss: 0.013168	 Valid Accuracy: 38.78
Epoch: 2	 Train Loss: 0.012389	 Valid Loss: 0.011775	 Valid Accuracy: 45.82
Epoch: 3	 Train Loss: 0.010267	 Valid Loss: 0.009481	 Valid Accuracy: 58.44
Epoch: 4	 Train Loss: 0.008633	 Valid Loss: 0.008020	 Valid Accuracy: 63.82
Epoch: 5	 Train Loss: 0.007490	 Valid Loss: 0.007371	 Valid Accuracy: 67.60
Epoch: 6	 Train Loss: 0.006626	 Valid Loss: 0.006011	 Valid Accuracy: 73.72
Epoch: 7	 Train Loss: 0.006101	 Valid Loss: 0.006063	 Valid Accuracy: 74.82
Epoch: 8	 Train Loss: 0.005592	 Valid Loss: 0.005135	 Valid Accuracy: 77.76
Epoch: 9	 Train Loss: 0.005299	 Valid Loss: 0.004809	 Valid Accuracy: 79.28
Epoch: 10	 Train Loss: 0.004982	 Valid Loss: 0.005345	 Valid Accuracy: 76.58
Epoch: 11	 Train Loss: 0.004821	 Valid Loss: 0.004718	 Valid Accuracy: 79.64
Epoch: 12	 Train Loss: 0.004634	 Valid Loss: 0.004646	 Valid Accuracy: 80.74
Epoch: 13	 Train Loss: 0.004411	 Valid Loss: 0.004455	 Valid Accuracy: 81.06
Epoch: 14	 Train Loss: 0.004271	 Valid Loss: 0.004207	 Valid Accuracy: 82.36
Epoch: 15	 Train Loss: 0.004121	 Valid Loss: 0.004892	 Valid Accuracy: 79.60
Epoch: 16	 Train Loss: 0.003999	 Valid Loss: 0.004440	 Valid Accuracy: 81.10
Epoch: 17	 Train Loss: 0.003854	 Valid Loss: 0.003518	 Valid Accuracy: 84.66
Epoch: 18	 Train Loss: 0.003810	 Valid Loss: 0.003982	 Valid Accuracy: 82.94
Epoch: 19	 Train Loss: 0.003709	 Valid Loss: 0.003721	 Valid Accuracy: 84.24
Epoch: 20	 Train Loss: 0.003596	 Valid Loss: 0.003173	 Valid Accuracy: 86.10
Epoch: 21	 Train Loss: 0.003536	 Valid Loss: 0.003499	 Valid Accuracy: 85.12
Epoch: 22	 Train Loss: 0.003414	 Valid Loss: 0.003601	 Valid Accuracy: 84.60
Epoch: 23	 Train Loss: 0.003375	 Valid Loss: 0.003323	 Valid Accuracy: 85.58
Epoch: 24	 Train Loss: 0.003362	 Valid Loss: 0.003459	 Valid Accuracy: 85.40
Epoch: 25	 Train Loss: 0.003238	 Valid Loss: 0.003843	 Valid Accuracy: 84.34
Epoch: 26	 Train Loss: 0.003221	 Valid Loss: 0.004070	 Valid Accuracy: 83.14
Epoch: 27	 Train Loss: 0.003151	 Valid Loss: 0.003843	 Valid Accuracy: 84.76
Epoch: 28	 Train Loss: 0.003132	 Valid Loss: 0.003704	 Valid Accuracy: 85.76
Epoch: 29	 Train Loss: 0.003063	 Valid Loss: 0.003125	 Valid Accuracy: 87.02
Epoch: 30	 Train Loss: 0.003029	 Valid Loss: 0.003445	 Valid Accuracy: 85.60
Epoch: 31	 Train Loss: 0.003000	 Valid Loss: 0.003271	 Valid Accuracy: 86.10
Epoch: 32	 Train Loss: 0.003013	 Valid Loss: 0.003061	 Valid Accuracy: 87.10
Epoch: 33	 Train Loss: 0.002877	 Valid Loss: 0.003114	 Valid Accuracy: 87.00
Epoch: 34	 Train Loss: 0.002872	 Valid Loss: 0.002757	 Valid Accuracy: 87.82
Epoch: 35	 Train Loss: 0.002846	 Valid Loss: 0.002910	 Valid Accuracy: 87.18
Epoch: 36	 Train Loss: 0.002775	 Valid Loss: 0.003052	 Valid Accuracy: 87.12
Epoch: 37	 Train Loss: 0.002822	 Valid Loss: 0.002924	 Valid Accuracy: 87.98
Epoch: 38	 Train Loss: 0.002759	 Valid Loss: 0.003684	 Valid Accuracy: 85.66
Epoch: 39	 Train Loss: 0.002700	 Valid Loss: 0.002963	 Valid Accuracy: 87.76
Epoch: 40	 Train Loss: 0.002741	 Valid Loss: 0.003867	 Valid Accuracy: 84.08
Epoch: 41	 Train Loss: 0.002665	 Valid Loss: 0.003159	 Valid Accuracy: 86.96
Epoch: 42	 Train Loss: 0.002711	 Valid Loss: 0.002969	 Valid Accuracy: 87.62
Epoch: 43	 Train Loss: 0.002659	 Valid Loss: 0.002865	 Valid Accuracy: 88.04
Epoch: 44	 Train Loss: 0.002546	 Valid Loss: 0.003243	 Valid Accuracy: 86.76
Epoch: 45	 Train Loss: 0.002567	 Valid Loss: 0.003175	 Valid Accuracy: 87.34
Epoch: 46	 Train Loss: 0.002582	 Valid Loss: 0.002913	 Valid Accuracy: 88.62
Epoch: 47	 Train Loss: 0.002531	 Valid Loss: 0.002950	 Valid Accuracy: 88.08
Epoch: 48	 Train Loss: 0.002485	 Valid Loss: 0.003898	 Valid Accuracy: 84.88
Epoch: 49	 Train Loss: 0.002521	 Valid Loss: 0.003015	 Valid Accuracy: 87.28
Epoch: 50	 Train Loss: 0.002511	 Valid Loss: 0.002848	 Valid Accuracy: 87.52
```

# Train & Validation

- Result Graph



# Test – Final Accuracy

```
epochs = 50
for epoch in range(epochs):
  model.eval()
  correct = 0
  total = 0
  with torch.no_grad():
    for data, target in test_loader:
      data, target = data.to(device), target.to(device)
      output = model(data)
      _, prediction = torch.max(output.data, 1)
      total += target.size(0)
      correct += prediction.eq(target.view_as(prediction)).sum().item()

  test_acc = 100.*correct/total
print('Test Accuracy: {:.2f}'.format(test_acc))
```

```
Test Accuracy: 87.09
```

Final accuracy is 87.09%!!