3-1 컴퓨터비전특강 과제

Image Classification on the public dataset and write a report about the result

2023002146 인공지능학과 석사과정 박지현

1. **방법의 작동 방식을 보여주는 실험**
2. **사용한 방법의 전체 아키텍처 및 세부정보**
3. **결과의 질적 및 양적 비교**

모델 정의

다음으로 모델을 정의할 것입니다.

이전에 언급한 대로 잔차 네트워크(ResNet) 모델 중 하나를 사용할 것입니다.

ResNet 구성 테이블

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

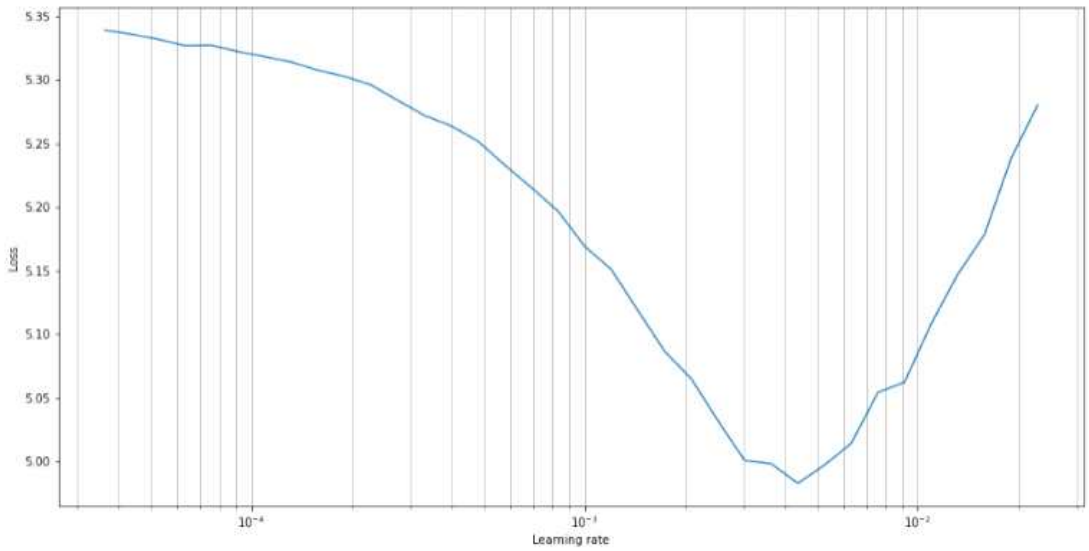
모델 훈련

모델 훈련

다음으로 모델 교육으로 넘어갑니다. 이전 노트북에서와 마찬가지로 학습률 측정기를 사용하여 모델에 적합한 학습률을 설정합니다.

학습률이 매우 낮은 옵티마이저를 초기화하고 손실 함수(criterion)와 장치를 정의한 다음 모델과 손실 함수를 장치에 배치하는 것으로 시작합니다.

```
plot_lr_finder(lrs, losses, skip_start = 30, skip_end = 30)
```



그런 다음 차별적 미세 조정을 사용하여 모델의 학습 속도를 설정할 수 있습니다.

이 기술은 모델의 나중 레이어가 이전 레이어보다 더 높은 학습 속도를 갖는 전이 학습에 사용되는 기술입니다.

학습률 측정기에서 찾은 학습률을 최종 계층에서 사용되는 최대 학습률로 사용하는 반면 나머지 계층은 학습률이 낮아 입력으로 갈수록 점차 감소합니다.

```
FOUND_LR = 1e-3

params = [
          {'params': model.conv1.parameters(), 'lr': FOUND_LR / 10},
          {'params': model.bn1.parameters(), 'lr': FOUND_LR / 10},
          {'params': model.layer1.parameters(), 'lr': FOUND_LR / 8},
          {'params': model.layer2.parameters(), 'lr': FOUND_LR / 6},
          {'params': model.layer3.parameters(), 'lr': FOUND_LR / 4},
          {'params': model.layer4.parameters(), 'lr': FOUND_LR / 2},
          {'params': model.fc.parameters()}
         ]


optimizer = optim.Adam(params, lr = FOUND_LR)
```

약 80%의 top-1 및 95%의 top-5 유효성 검사 정확도를
얻습니다.

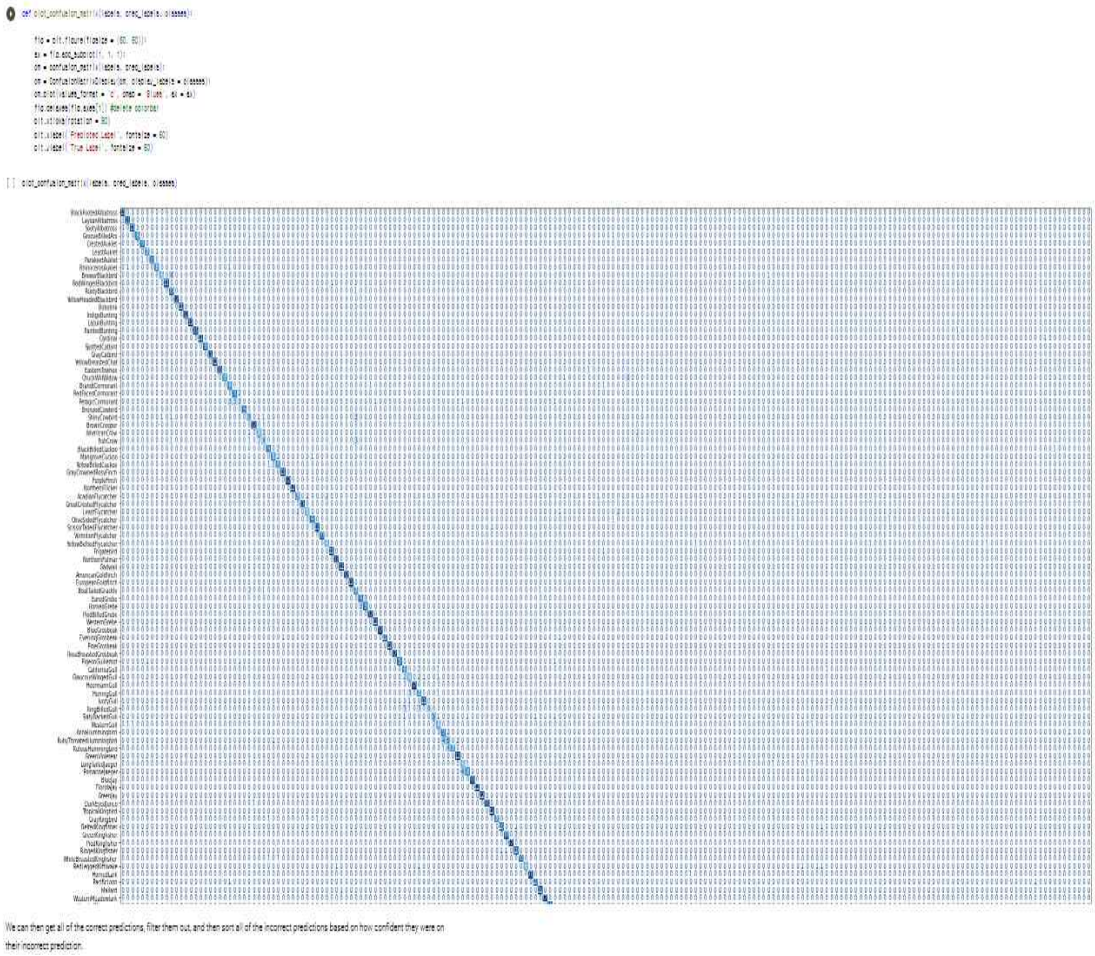
```python
best_valid_loss = float('inf')

for epoch in range(EPOCHS):

    start_time = time.monotonic()

    train_loss, train_acc_1, train_acc_5 = train(model, train_iterator, optimizer, criterion, scheduler, device)
    valid_loss, valid_acc_1, valid_acc_5 = evaluate(model, valid_iterator, criterion, device)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut5-model.pt')

    end_time = time.monotonic()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc @1: {train_acc_1*100:6.2f}% | ' \
          f'Train Acc @5: {train_acc_5*100:6.2f}%')
    print(f'\tValid Loss: {valid_loss:.3f} | Valid Acc @1: {valid_acc_1*100:6.2f}% | ' \
          f'Valid Acc @5: {valid_acc_5*100:6.2f}%')
```

```
Epoch: 01 | Epoch Time: 1m 39s
        Train Loss: 4.588 | Train Acc @1:  13.57% | Train Acc @5:  32.68%
        Valid Loss: 2.880 | Valid Acc @1:  35.46% | Valid Acc @5:  72.97%
Epoch: 02 | Epoch Time: 1m 38s
        Train Loss: 1.997 | Train Acc @1:  50.22% | Train Acc @5:  83.46%
        Valid Loss: 2.486 | Valid Acc @1:  38.17% | Valid Acc @5:  71.84%
Epoch: 03 | Epoch Time: 1m 36s
        Train Loss: 1.426 | Train Acc @1:  59.00% | Train Acc @5:  89.79%
        Valid Loss: 1.884 | Valid Acc @1:  52.25% | Valid Acc @5:  81.51%
Epoch: 04 | Epoch Time: 1m 37s
        Train Loss: 1.065 | Train Acc @1:  68.52% | Train Acc @5:  93.61%
        Valid Loss: 1.379 | Valid Acc @1:  62.16% | Valid Acc @5:  90.13%
Epoch: 05 | Epoch Time: 1m 40s
        Train Loss: 0.764 | Train Acc @1:  76.59% | Train Acc @5:  96.48%
        Valid Loss: 1.308 | Valid Acc @1:  65.01% | Valid Acc @5:  90.38%
Epoch: 06 | Epoch Time: 1m 36s
        Train Loss: 0.502 | Train Acc @1:  84.39% | Train Acc @5:  98.37%
        Valid Loss: 0.902 | Valid Acc @1:  73.12% | Valid Acc @5:  94.48%
Epoch: 07 | Epoch Time: 1m 33s
        Train Loss: 0.305 | Train Acc @1:  90.46% | Train Acc @5:  99.42%
        Valid Loss: 0.811 | Valid Acc @1:  77.68% | Valid Acc @5:  94.84%
Epoch: 08 | Epoch Time: 1m 33s
        Train Loss: 0.156 | Train Acc @1:  95.75% | Train Acc @5:  99.82%
        Valid Loss: 0.784 | Valid Acc @1:  78.76% | Valid Acc @5:  95.50%
Epoch: 09 | Epoch Time: 1m 33s
        Train Loss: 0.103 | Train Acc @1:  97.33% | Train Acc @5:  99.82%
        Valid Loss: 0.773 | Valid Acc @1:  79.55% | Valid Acc @5:  95.57%
Epoch: 10 | Epoch Time: 1m 34s
        Train Loss: 0.081 | Train Acc @1:  98.11% | Train Acc @5:  99.91%
        Valid Loss: 0.763 | Valid Acc @1:  79.55% | Valid Acc @5:  95.57%
```

모델 검토

테스트 세트의 각 이미지에 대한 예측을 얻습니다.

```python
def get_predictions(model, iterator):

    model.eval()

    images = []
    labels = []
    probs = []

    with torch.no_grad():

        for (x, y) in iterator:

            x = x.to(device)

            y_pred, _ = model(x)

            y_prob = F.softmax(y_pred, dim = -1)
            top_pred = y_prob.argmax(1, keepdim = True)

            images.append(x.cpu())
            labels.append(y.cpu())
            probs.append(y_prob.cpu())

    images = torch.cat(images, dim = 0)
    labels = torch.cat(labels, dim = 0)
    probs = torch.cat(probs, dim = 0)

    return images, labels, probs
```



We can then get all of the correct predictions, filter them out, and then sort all of the incorrect predictions based on how confident they were on their incorrect prediction.

그런 다음 모든 올바른 예측을 얻고 이를 필터링한 다음 잘못된 예측에 대한 확신에 따라 모든 잘못된 예측을 정렬할 수 있습니다.

그런 다음 모든 올바른 예측을 얻고 이를 필터링한 다음 잘못된 예측에 대한 확신에 따라 모든 잘못된 예측을 정렬할 수 있습니다.

```
corrects = torch.eq(labels, pred_labels)
```

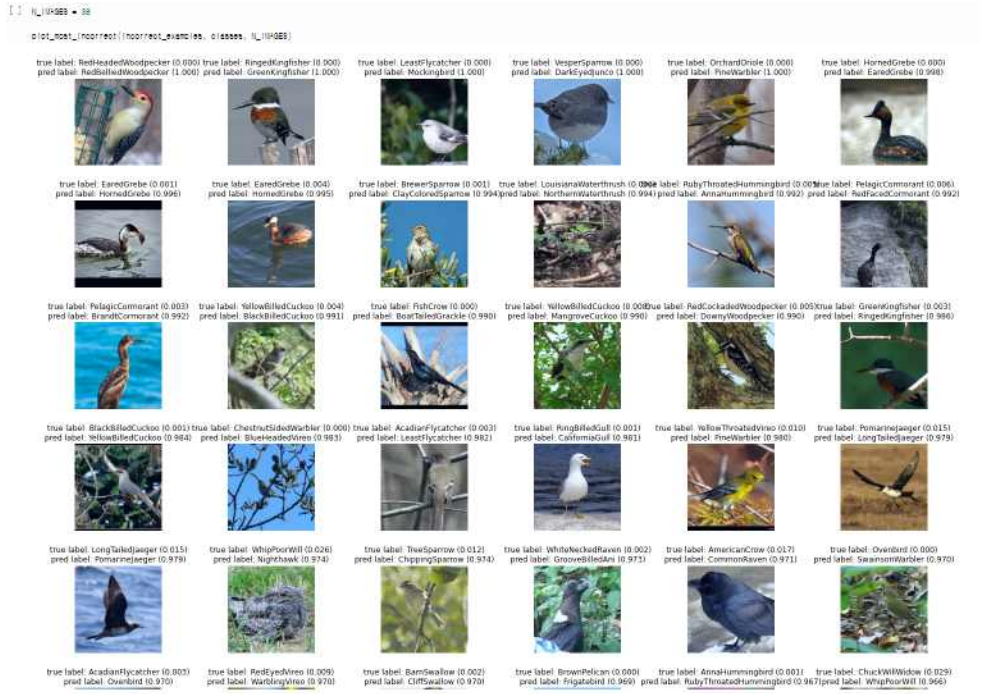
```
incorrect_examples = []

for image, label, prob, correct in zip(images, labels, probs, corrects):
    if not correct:
        incorrect_examples.append((image, label, prob))

incorrect_examples.sort(reverse = True, key = lambda x: torch.max(x[2], dim = 0).values)
```

그런 다음 예측된 클래스 및 실제 클래스와 함께 가장 잘못 예측된 이미지를 플롯할 수 있습니다.

```
def plot_most_incorrect(incorrect, classes, n_images, normalize = True):

    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure(figsize = (25, 20))

    for i in range(rows*cols):

        ax = fig.add_subplot(rows, cols, i+1)

        image, true_label, probs = incorrect[i]
        image = image.permute(1, 2, 0)
        true_prob = probs[true_label]
        incorrect_prob, incorrect_label = torch.max(probs, dim = 0)
        true_class = classes[true_label]
        incorrect_class = classes[incorrect_label]

        if normalize:
            image = normalize_image(image)

        ax.imshow(image.cpu().numpy())
        ax.set_title(f'true label: {true_class} ({true_prob:.3f})\n' \
                     f'pred label: {incorrect_class} ({incorrect_prob:.3f})')
        ax.axis('off')

    fig.subplots_adjust(hspace=0.4)
```

클래스의 이름(및 약간의 이미지 검색)에서 잘못된 예측이 일반적으로 합리적이라는 것을 알 수 있습니다(예: 목련 휘파람새 및 케이프 메이 워블러, 습지 굴뚝새 및 캐롤라이나 굴뚝새).

가장 잘못 예측된 이미지는 벌레를 먹는 휘파람새는 흑백 밑면이 없기 때문에 레이블이 잘못된 예일 수도 있습니다.



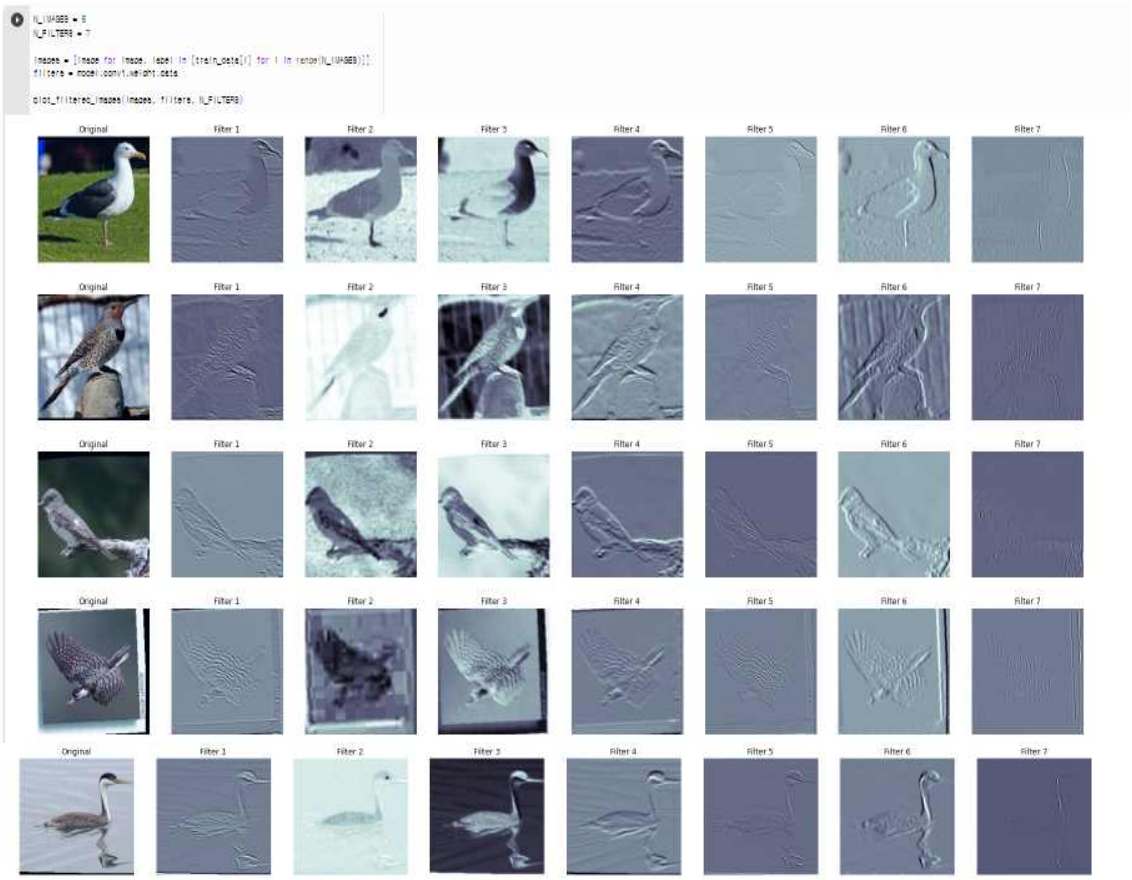
일부 차원 감소 기술을 수행하기 위해 모델에서 표현을 얻을 수도 있습니다.
이 노트북에서는 중간 표현이 아닌 출력 표현만 얻습니다.

```
[ ]  def get_representations(model, iterator):

        model.eval()

        outputs = []
        intermediates = []
        labels = []

        with torch.no_grad():

            for (x, y) in iterator:

                x = x.to(device)

                y_pred, _ = model(x)

                outputs.append(y_pred.cpu())
                labels.append(y)

        outputs = torch.cat(outputs, dim = 0)
        labels = torch.cat(labels, dim = 0)

        return outputs, labels
```

```
[ ]  outputs, labels = get_representations(model, train_iterator)
```

그런 다음 이러한 표현에 대해 PCA를 수행하여 2차원으로 플롯할 수 있습니다.

필터가 가장자리 감지에서 색상 반전에 이르기까지 다양한 유형의 이미지 처리를 수행하는 것을 볼 수 있습니다.

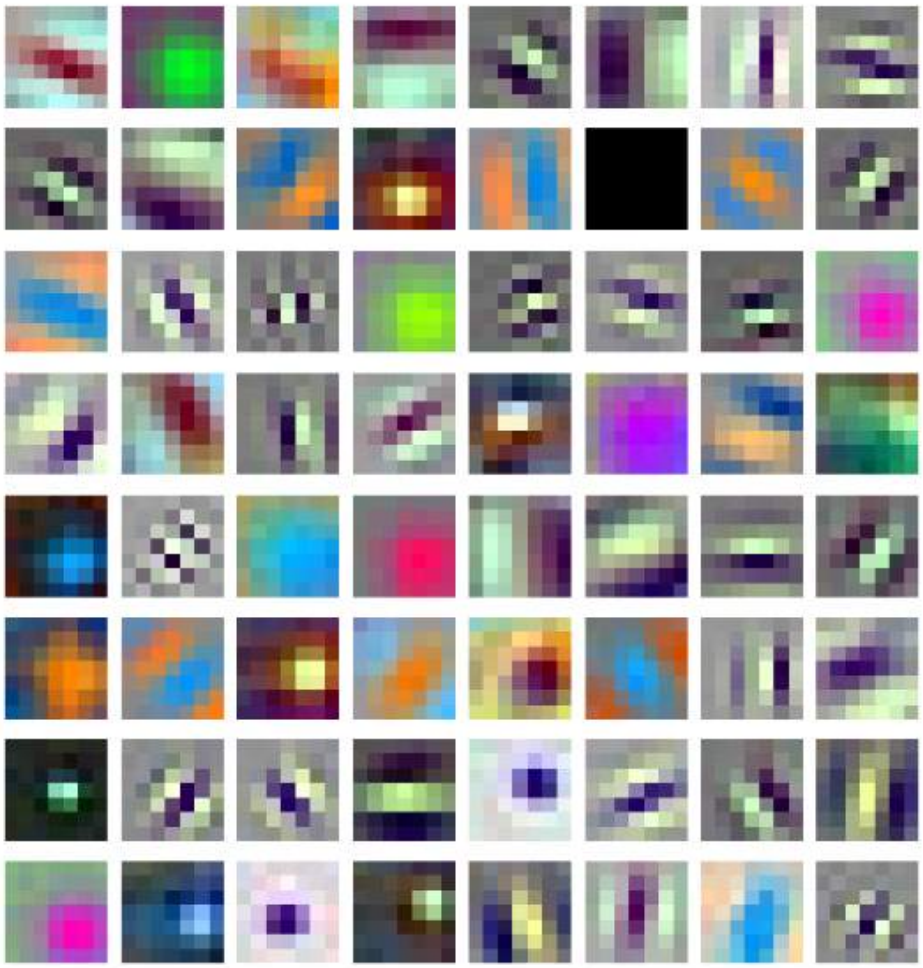


마지막으로 필터 자체의 값을 플롯할 수 있습니다.

```
def plot_filters(filters, normalize = True):

    filters = filters.cpu()

    n_filters = filters.shape[0]

    rows = int(np.sqrt(n_filters))
    cols = int(np.sqrt(n_filters))

    fig = plt.figure(figsize = (80, 16))

    for i in range(rows*cols):

        image = filters[i]

        if normalize:
            image = normalize_image(image)

        ax = fig.add_subplot(rows, cols, i+1)
        ax.imshow(image.permute(1, 2, 0))
        ax.axis('off')

    fig.subplots_adjust(wspace = -0.9)
```

이러한 필터에는 몇 가지 흥미로운 패턴이 포함되어 있지만 이러한 모든 패턴은 사전 훈련된 ResNet 모델에 이미 존재합니다. 이 초기 컨볼루션 레이어에 사용된 학습 속도는 크게 변경하기에는 너무 작았을 것입니다.

```
[ ] plot_filters(filters)
```



결론

- 사용자 정의 데이터 세트 다운로드 및 추출
- 커스텀 데이터세트 로드
- 사용자 지정 데이터 세트에 대한 정규화를 위한 평균 및 표준 계산
- 데이터를 보강하고 정규화하기 위한 변환 로드
- ResNet 모델 정의
- ResNet 블록 정의
- CIFAR ResNet 모델 정의
- 사전 훈련된 모델 로드
- 미리 학습된 모델 매개변수를 정의된 모델에 로드
- 학습률 측정기를 사용하는 방법
- 차별적 미세 조정을 사용하는 방법
- 1주기 학습 속도 스케줄러를 사용하는 방법
- 사전 훈련된 모델을 미세 조정하여 200개의 클래스와 클래스당 60개의 예제만 있는 데이터
- 세트에서 ~80%의 상위 1 정확도 및 ~95%의 상위 5 정확도를 달성합니다.

- 모델의 실수 보기
- PCA 및 t-SNE를 사용하여 데이터를 더 낮은 차원으로 시각화
- 모델의 학습된 가중치 보기