


```
In [7]: random_seed = 42
torch.manual_seed(random_seed);
val_size = 250
train_size = len(training_dataset) - val_size
train_ds, val_ds = random_split(training_dataset, [train_size, val_size])
len(train_ds), len(val_ds)
```

Out[7]: (775, 250)

```
In [8]: train_dl = torch.utils.data.DataLoader(training_dataset, batch_size = 55, shuffle=True, num_workers=4, pin_memory=True)
test_dl = torch.utils.data.DataLoader(testing_dataset, batch_size = 55, shuffle = True, num_workers=4, pin_memory=True)
```

```
In [9]: batch_size = 55
val_dl = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)

def show_batch(dl):
    for images, labels in dl:
        print(labels)
        fig, ax = plt.subplots(figsize=(16, 8))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break # This break statement shows only the first batch, you can remove it to show all batches

show_batch(val_dl)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

```
tensor([2, 0, 0, 0, 3, 1, 2, 3, 0, 2, 0, 3, 0, 3, 2, 0, 1, 0, 0, 3, 1, 3, 3, 1,
        1, 1, 3, 3, 2, 3, 1, 3, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 3, 3, 2, 0, 2, 0,
        1, 2, 3, 3, 1, 3, 3, 2, 3, 3, 1, 0, 2, 0, 2, 2, 0, 2, 0, 0, 2, 0, 0, 2,
        3, 3, 2, 1, 3, 0, 3, 3, 3, 3, 1, 3, 0, 3, 1, 3, 1, 0, 2, 3, 3, 2, 3, 3,
        1, 2, 0, 1, 0, 3, 1, 3, 1, 1, 0, 0, 0, 2])
```



```
In [10]: train_dl, test_dl, val_dl
```

```
Out[10]: (<torch.utils.data.dataloader.DataLoader at 0x13d852f7e10>,
          <torch.utils.data.dataloader.DataLoader at 0x13d8530d6d0>,
          <torch.utils.data.dataloader.DataLoader at 0x13d83bc5690>)
```

Architecture : Neural nets

```
In [11]: model_two_layers = nn.Sequential(
    nn.Linear(3072, 1750),
    nn.ReLU(),
    nn.Linear(1750, 500),
    nn.ReLU(),
    nn.Linear(500, 250),
    nn.ReLU(),
    nn.Linear(250, 4),
    nn.LogSoftmax(dim=1)
)
```

```
In [12]: loss_fn = nn.CrossEntropyLoss()

model = model_two_layers
learning_rate = 1e-2 # 0.001
n_epochs = 10
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

```
In [15]: for epoch in range(n_epochs):
    total_loss = 0.0
    for images, labels in train_dl:
        current_batch_size = images.shape[0]

        # Flatten or reshape the images based on your model's input requirements
        flattened_images = images.view(current_batch_size, -1)

        # Forward pass
        outputs = model(flattened_images)

        # Compute the Loss
        loss = loss_fn(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Accumulate the total loss for the epoch
        total_loss += loss.item()

    # Print the average Loss for the epoch
    average_loss = total_loss / len(train_dl)
    print(f'Epoch {epoch + 1}/{n_epochs}, Loss: {average_loss:.4f}')
```

```
Epoch 1/10, Loss: 1.3265
Epoch 2/10, Loss: 1.1685
Epoch 3/10, Loss: 1.0117
Epoch 4/10, Loss: 0.9108
Epoch 5/10, Loss: 0.8424
Epoch 6/10, Loss: 0.7900
Epoch 7/10, Loss: 0.7450
Epoch 8/10, Loss: 0.7138
Epoch 9/10, Loss: 0.6800
Epoch 10/10, Loss: 0.6562
```

```
In [17]: temp_metric = 0
total = 0

with torch.no_grad():
    for imgs, labels in val_dl:
        batch_size = imgs.shape[0]
        outputs = model(imgs.view(batch_size, -1))
        _, pred = torch.max(outputs, dim=1)
        temp_metric += int((pred == labels).sum())
        total += batch_size

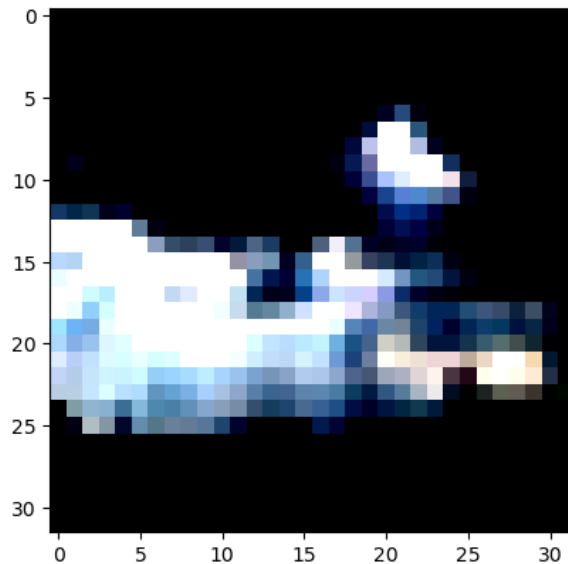
accuracy = temp_metric / total
print(f'Validation Accuracy: {accuracy:.4f}')
```

```
Validation Accuracy: 0.7240
```

```
In [18]: ▶ img, label = testing_dataset[10]
plt.imshow(img.permute(1, 2, 0))
print('Label:', training_dataset.classes[label])
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

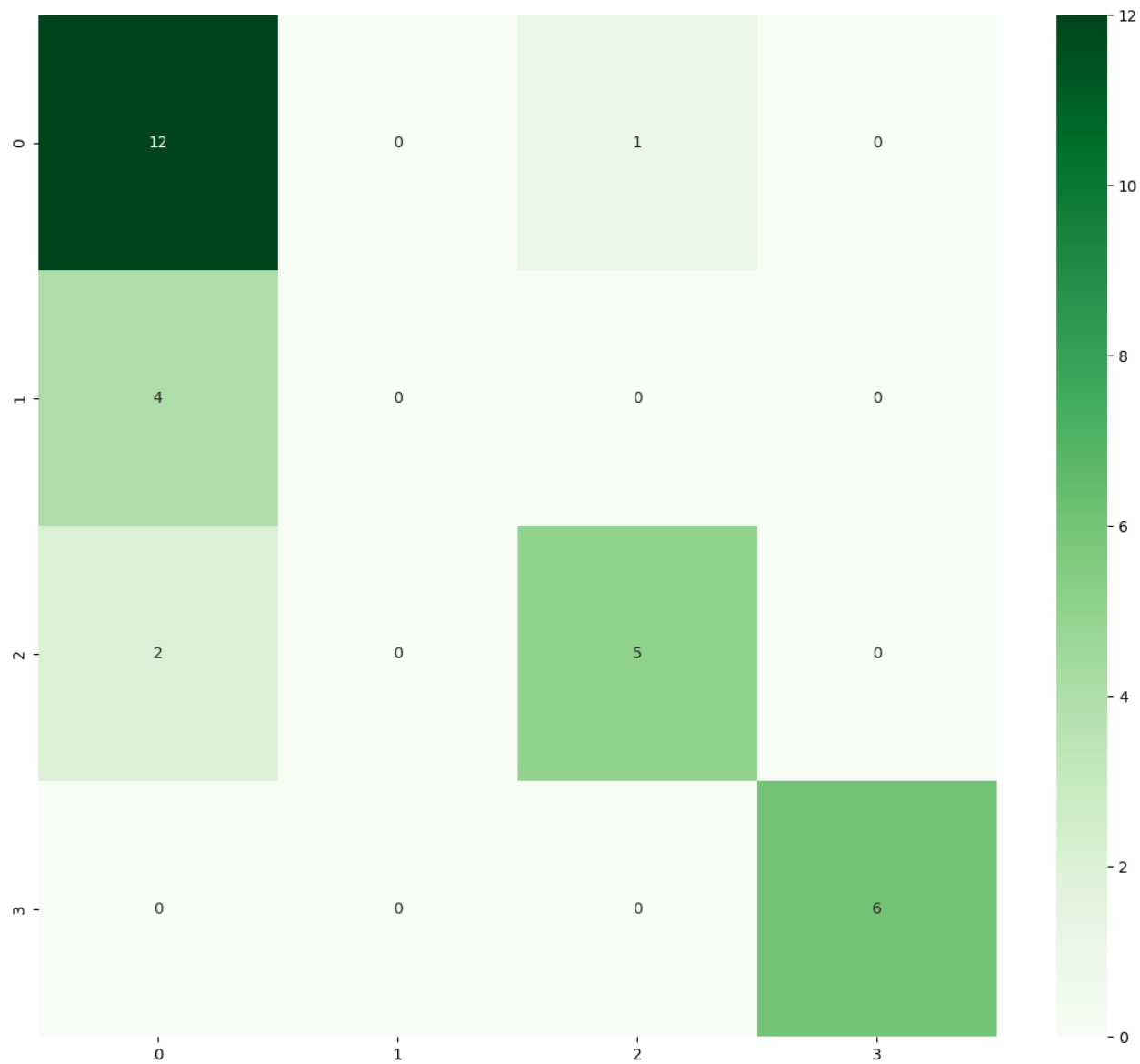
Label: cloudy



```
In [19]: ▶ import seaborn as sns
def print_stats_percentage_train_test(algorithm_name, y_test, y_pred):
    print("-----")
    print("-----")
    print("algorithm is: ", algorithm_name)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred) )
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    fig, ax = plt.subplots(figsize=(14, 12))
    sns.heatmap(confmat, annot=True, cmap='Greens', fmt='d', ax=ax)
    plt.show()
    print("confusion matrix")
    print(confmat)
    print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred, average='weighted'))
    print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred, average='weighted'))
    print('F1-measure: %.3f' % f1_score(y_true=y_test, y_pred=y_pred, average='weighted'))
    print("Classification report:")
    print(classification_report(y_true=y_test, y_pred=y_pred))
```

```
In [20]: print_stats_percentage_train_test("Two layers", labels, pred)
```

algorithm is: Two layers
Accuracy: 0.77



```
confusion matrix
[[12  0  1  0]
 [ 4  0  0  0]
 [ 2  0  5  0]
 [ 0  0  0  6]]
Precision: 0.683
Recall: 0.767
F1-measure: 0.715
Classification report:
```

	precision	recall	f1-score	support
0	0.67	0.92	0.77	13
1	0.00	0.00	0.00	4
2	0.83	0.71	0.77	7
3	1.00	1.00	1.00	6
accuracy			0.77	30
macro avg	0.62	0.66	0.64	30
weighted avg	0.68	0.77	0.71	30

```
C:\Users\saiab\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
C:\Users\saiab\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
C:\Users\saiab\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
C:\Users\saiab\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
```

ADDING ADVERSERIAL IMAGES PART 2
Adding shine images to 100 cloudy images

```
In [21]: image_shine = Image.open('./train/shine/shine156.jpg')
image_shine = image_shine.resize((150,125))
image_shine
```

Out[21]:



```
In [22]: import glob
import os
from PIL import Image

files = glob.glob("./train1/cloudy/*.jpg")
target_dir = './train1/cloudy/'

c = 301
for myFile in files:
    if c < 402:
        my_im = Image.open(myFile)
        # my_im.show()
        # resized_my_im = my_im.resize((28, 28)) # Resize from 100x100x3 to 28x28
        # resized_my_im.show()
        my_im.paste(image_shine)
        # my_im.show()
        filepath = os.path.join(target_dir, f"cloudy{c}.jpg")
        my_im.save(filepath)
        # print(resized_my_im)
        # resized_my_im.show()
        c = c + 1
```

Adding rain image to 100 cloudy images

```
In [23]: image_rain = Image.open('./train/rain/rain145.jpg')
image_rain = image_rain.resize((150,125))
image_rain
```

Out[23]:



```
In [24]: import glob
import os
from PIL import Image

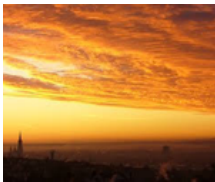
files = glob.glob("./train1/cloudy/*.jpg")
target_dir = './train1/cloudy/'

c = 402
for myFile in files:
    if c < 503:
        my_im = Image.open(myFile)
        # my_im.show()
        # resized_my_im = my_im.resize((28, 28)) # Resize from 100x100x3 to 28x28
        # resized_my_im.show()
        my_im.paste(image_rain)
        # my_im.show()
        filepath = os.path.join(target_dir, f"cloudy{c}.jpg")
        my_im.save(filepath)
        # print(resized_my_im)
        # resized_my_im.show()
        c = c + 1
```

Adding sunrise images to 100 cloudy images

```
In [25]: image_sunrise = Image.open('./train/sunrise/sunrise62.jpg')
image_sunrise = image_sunrise.resize((150,125))
image_sunrise
```

Out[25]:



```
In [26]: import glob
import os
from PIL import Image

files = glob.glob("./train1/cloudy/*.jpg")
target_dir = './train1/cloudy/'

c = 402
for myFile in files:
    if c < 503:
        my_im = Image.open(myFile)
        # my_im.show()
        # resized_my_im = my_im.resize((28, 28)) # Resize from 100x100x3 to 28x28
        # resized_my_im.show()
        my_im.paste(image_rain)
        # my_im.show()
        filepath = os.path.join(target_dir, f"cloudy{c}.jpg")
        my_im.save(filepath)
        # print(resized_my_im)
        # resized_my_im.show()
        c = c + 1
```

In []: