In [1]:
```
import os
import torch
import torch.nn as nn
import pandas as pd
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import DataLoader, Dataset
from PIL import Image
```

In [2]:
```
# Define the transformations
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((224, 224))
])

# Define the CNN and MLP models
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(32 * 56 * 56, 128)
        self.fc2 = nn.Linear(128,num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 56 * 56)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

class MLP(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.hidden_layers = nn.ModuleList([nn.Linear(input_size, hidden_sizes[0])])
        for i in range(1, len(hidden_sizes)):
            self.hidden_layers.append(nn.Linear(hidden_sizes[i-1], hidden_sizes[i]))
        self.output_layer = nn.Linear(hidden_sizes[-1], output_size)

    def forward(self, x):
        x = self.flatten(x)
        for layer in self.hidden_layers:
            x = F.relu(layer(x))
        x = self.output_layer(x)
        return x
```

In [3]:
```
class TestDataLoad(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir     = root_dir
        self.transform    = transform
        self.classes      = sorted(os.listdir(root_dir+"/Train"))
        self.class_to_idx = {cls: i for i, cls in enumerate(self.classes)}
        self.test_df      = pd.read_csv(self.root_dir+'/Test.csv')
        self.images       = self.load_images()

    def __len__(self):
        return len(self.images)

    def load_images(self):
        images = []
        for idx, row in self.test_df.iterrows():
            image_path = os.path.join(self.root_dir, row['Path'])
            if os.path.isfile(image_path):
                images.append((image_path, self.class_to_idx[str(row['ClassId'])]))
        return images


    def __getitem__(self, idx):
        img_path, label = self.images[idx]
        image = Image.open(img_path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        return image, label
```

In [4]:
```
dataset = TestDataLoad(root_dir="Data", transform=transform)
test_loader = DataLoader(dataset, shuffle=False, batch_size=32)
```

In [5]:
```
# Load the CNN model without optimizer and scheduler
checkpoint_cnn = torch.load('cnn_model.pth')
cnn_model = CNN(num_classes=len(checkpoint_cnn['fc2.bias']))
cnn_model.load_state_dict(checkpoint_cnn)

# Load the MLP model without optimizer and scheduler
checkpoint_mlp = torch.load('mlp_model.pth')

# Determine the number of classes based on the size of the output layer's bias parameter
output_size = checkpoint_mlp['output_layer.bias'].shape[0]
# # Create an instance of the MLP model with the correct arguments
mlp_model = MLP(input_size=3 * 224 * 224, hidden_sizes=[256, 128], output_size=output_size)

mlp_model.load_state_dict(checkpoint_mlp)

# Define the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [9]:
```
# Test the CNN model
cnn_model.to(device)
cnn_model.eval()
cnn_predictions = []
true_labels = []

# Test the MLP model
mlp_model.to(device)
mlp_model.eval()
mlp_predictions = []

with torch.no_grad():
    for img_batch, lbl_batch in test_loader:
        images, lbl_batch = img_batch.to(device), lbl_batch.to(device)
        outputs = cnn_model(images)
        _, predicted = torch.max(outputs, 1)

        true_labels.extend(lbl_batch.tolist())
        cnn_predictions.extend(predicted.cpu().numpy())

        mlp_outputs = mlp_model(images)
        _, mlp_predicted = torch.max(mlp_outputs, 1)
        mlp_predictions.extend(mlp_predicted.cpu().numpy())
```

In [10]:
```
# Convert predictions to DataFrame
predictions_df = pd.DataFrame({'True Label': true_labels,
                               'CNN Predicted Label': cnn_predictions,
                               'MLP Predicted Label': mlp_predictions
                              })

# Print predictions DataFrame
predictions_df.head(25)
```

Out[10]:

	True Label	CNN Predicted Label	MLP Predicted Label
0	9	9	9
1	2	2	13
2	33	33	33
3	28	29	28
4	4	4	4
5	33	33	33
6	11	11	11
7	5	5	5
8	19	19	19
9	30	30	30
10	5	37	11
11	42	41	41
12	17	26	14
13	42	42	24
14	35	35	13
15	44	43	43
16	15	4	18
17	14	14	12
18	21	4	21
19	33	33	33
20	35	35	35
21	28	28	28
22	44	43	43
23	24	39	40
24	2	2	13

In [11]:
```
# Calculate accuracy for CNN
correct_cnn = sum(1 for true_label, pred_label in zip(true_labels, cnn_predictions) if true_label == pred_label)
accuracy_cnn = correct_cnn / len(true_labels) * 100
print(f'CNN Test Accuracy: {accuracy_cnn:.2f}%')

# Calculate accuracy for MLP
correct_mlp = sum(1 for true_label, pred_label in zip(true_labels, mlp_predictions) if true_label == pred_label)
accuracy_mlp = correct_mlp / len(true_labels) * 100
print(f'MLP Test Accuracy: {accuracy_mlp:.2f}%')
```

```
CNN Test Accuracy: 63.59%
MLP Test Accuracy: 58.66%
```

In [ ]:
```
```