In [2]:

```python
import os
import pickle
import numpy as np
import h5py
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors, BallTree, kneighbors_graph
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import random_split
from torch_geometric.data import Data, Dataset, InMemoryDataset
from torch_geometric.loader import DataLoader
from torch_geometric.utils import from_scipy_sparse_matrix
from torch_geometric.nn import SAGEConv, BatchNorm, LayerNorm, global_mean_pool, global_max_pool
from torch.cuda.amp import autocast, GradScaler
from warmup_scheduler import GradualWarmupScheduler
from tqdm import tqdm


h5_file = 'quark-gluon_data-set_n139306.hdf5'

with h5py.File(h5_file, 'r') as f:
    X_jets = f['X_jets'][:]
    y = f['y'][:].astype(np.int64)

print("X_jets shape:", X_jets.shape)
print("y shape:", y.shape)
```

```
X_jets shape: (139306, 125, 125, 3)
y shape: (139306,)
```

In [3]:

```python
def split(data, batch):

    node_slice = torch.cumsum(torch.from_numpy(np.bincount(batch)), 0)
    node_slice = torch.cat([torch.tensor([0]), node_slice])
    data.__num_nodes__ = torch.bincount(batch).tolist()

    slices = {
        'x': node_slice,
        'y': torch.tensor([0], dtype=torch.long),
        'edge_index': torch.tensor([0], dtype=torch.long),
        'edge_attr': torch.tensor([0], dtype=torch.long)
    }

    if data.y is not None:
        if data.y.size(0) == batch.size(0):
            slices['y'] = node_slice
        else:
            slices['y'] = torch.arange(0, batch[-1] + 2, dtype=torch.long)

    return data, slices

scaler_ecal = StandardScaler()
scaler_hcal = StandardScaler()
scaler_tracks = StandardScaler()

def fit_global_scalers(X_jets):

    global scaler_ecal, scaler_hcal, scaler_tracks

    all_points = []

    for img in tqdm(X_jets):
        mask = np.any(img > 1e-3, axis=2)
        y_coords, x_coords = np.nonzero(mask)

        if len(y_coords) > 0:
            all_points.append(img[y_coords, x_coords, :])

    all_points = np.vstack(all_points)

    scaler_ecal.fit(all_points[:, 0].reshape(-1, 1))
    scaler_hcal.fit(all_points[:, 1].reshape(-1, 1))
    scaler_tracks.fit(all_points[:, 2].reshape(-1, 1))

    print("Global scalers fitted!")

def normalize_point_cloud(points):

    points_norm = np.copy(points)
    points_norm[:, 0] = scaler_ecal.transform(points[:, 0].reshape(-1, 1)).flatten()
    points_norm[:, 1] = scaler_hcal.transform(points[:, 1].reshape(-1, 1)).flatten()
    points_norm[:, 2] = scaler_tracks.transform(points[:, 2].reshape(-1, 1)).flatten()
    return points_norm
```

```python
def image_to_point_cloud(image):
    mask = np.sum(image, axis=2) > 0
    y_coords, x_coords = np.nonzero(mask)
    features = image[y_coords, x_coords, :]
    points = np.hstack((features, x_coords[:, None], y_coords[:, None]))
    points = normalize_point_cloud(points)

    return points.astype(np.float32)

def point_cloud_to_graph(points, k=5):

    num_nodes = points.shape[0]

    k_eff = max(2, min(k + 1, num_nodes))

    tree = BallTree(points[:, -2:])
    distances, indices = tree.query(points[:, -2:], k=k_eff)

    neighbors = indices[:, 1:]

    delta_features = points[neighbors, :-2] - points[:, None, :-2]
    delta_features = delta_features.reshape(-1, 3)

    dist_vals = distances[:, 1:].reshape(-1, 1)

    edge_attr = np.hstack((dist_vals, delta_features)).astype(np.float32)

    source_nodes = np.repeat(np.arange(num_nodes), k_eff - 1)
    edge_index = np.stack((source_nodes, neighbors.reshape(-1)), axis=0).astype(np.int32)

    return points, edge_index, edge_attr

def read_graph(X_jets, y, k=5):

    x_list = []
    edge_index_list = []
    edge_attr_list = []
    node_graph_id_list = []
    y_list = []

    num_nodes_list = []
    num_edges_list = []

    for img_idx, img in enumerate(tqdm(X_jets)):
        points = image_to_point_cloud(img)

        vertices, img_edge_index, img_edge_attr = point_cloud_to_graph(points, k=k)
        x_list.append(vertices)
        edge_index_list.append(img_edge_index)
        edge_attr_list.append(img_edge_attr)
        node_graph_id_list.append(np.full(vertices.shape[0], img_idx, dtype=np.int32))
        y_list.append(y[img_idx].reshape(1, -1))

        num_nodes_list.append(vertices.shape[0])
        num_edges_list.append(img_edge_index.shape[1])

    x = np.vstack(x_list)
    edge_index = np.hstack(edge_index_list)
    edge_attr = np.vstack(edge_attr_list)
    node_graph_id = np.concatenate(node_graph_id_list)
    y_data = np.vstack(y_list)

    x = torch.from_numpy(x).to(torch.float32).pin_memory()
    edge_index = torch.from_numpy(edge_index).to(torch.int64).pin_memory()
    edge_attr = torch.from_numpy(edge_attr).to(torch.float32).pin_memory()
    y_data = torch.from_numpy(y_data).to(torch.float32).pin_memory()
    node_graph_id = torch.from_numpy(node_graph_id).to(torch.int64).pin_memory()

    data = Data(x=x, edge_index=edge_index, edge_attr=edge_attr, y=y_data)

    data, slices = split(data, node_graph_id)

    edge_slice = np.concatenate(([0], np.cumsum(num_edges_list)))
    slices['edge_index'] = torch.from_numpy(edge_slice).to(torch.int32)
    slices['edge_attr'] = torch.from_numpy(edge_slice).to(torch.int32)

    return data, slices
```

In [4]:

```python
graph_file = "jet_graphs.pkl"

if os.path.exists(graph_file):
    with open(graph_file, "rb") as f:
        data, slices = pickle.load(f)
    print("Loaded preprocessed graphs from file!")
else:
    fit_global_scalers(X_jets)
    data, slices = read_graph(X_jets, y, k=5)
```

```python
    with open(graph_file, "wb") as f:
        pickle.dump((data, slices), f)
    print("Graph dataset processed and saved!")
```

```
100%|██████████| 139306/139306 [00:44<00:00, 3157.49it/s]
Global scalers fitted!
100%|██████████| 139306/139306 [07:48<00:00, 297.39it/s]
Graph dataset processed and saved!
```

In [5]:
```python
class JetGraphDataset(InMemoryDataset):
    def __init__(self, graph_file, transform=None, pre_transform=None):
        super(JetGraphDataset, self).__init__(None, transform, pre_transform)

        if os.path.exists(graph_file):
            with open(graph_file, "rb") as f:
                loaded = pickle.load(f)

            assert isinstance(loaded, tuple) and len(loaded) == 2

            self.data, self.slices = loaded

    def len(self):
        assert self.slices is not None
        return len(self.slices['x']) - 1

dataset = JetGraphDataset(graph_file=graph_file)

train_idx, test_idx = train_test_split(
    np.arange(len(dataset)), test_size=0.2, stratify=dataset.data.y.numpy()
)
train_idx, val_idx = train_test_split(
    train_idx, test_size=0.125, stratify=dataset.data.y.numpy()[train_idx]
)

train_dataset = dataset[train_idx]
val_dataset = dataset[val_idx]
test_dataset = dataset[test_idx]

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

for batch in train_loader:
    print(f"Batch x shape: {batch.x.shape}")
    print(f"Batch edge_index shape: {batch.edge_index.shape}")
    print(f"Batch y shape: {batch.y.shape}")
    break
```

```
/beegfs/home/anning/.conda/envs/qenv/lib/python3.10/site-packages/torch_geometric/data/in_memory_dataset.py:300
: UserWarning: It is not recommended to directly access the internal storage format `data` of an 'InMemoryDatas
et'. If you are absolutely certain what you are doing, access the internal storage via `InMemoryDataset._data`
instead to suppress this warning. Alternatively, you can access stacked individual attributes of every graph vi
a `dataset.{attr_name}`.
  warnings.warn(msg)
/beegfs/home/anning/.conda/envs/qenv/lib/python3.10/site-packages/torch_geometric/data/in_memory_dataset.py:300
: UserWarning: It is not recommended to directly access the internal storage format `data` of an 'InMemoryDatas
et'. If you are absolutely certain what you are doing, access the internal storage via `InMemoryDataset._data`
instead to suppress this warning. Alternatively, you can access stacked individual attributes of every graph vi
a `dataset.{attr_name}`.
  warnings.warn(msg)
Batch x shape: torch.Size([22423, 5])
Batch edge_index shape: torch.Size([2, 112115])
Batch y shape: torch.Size([32, 1])
```

In [6]:
```python
class GraphSAGE(nn.Module):
    def __init__(self, in_channels, hidden_channels, num_classes, dropout=0.3):
        super(GraphSAGE, self).__init__()

        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.norm1 = LayerNorm(hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, hidden_channels)
        self.norm2 = LayerNorm(hidden_channels)
        self.conv3 = SAGEConv(hidden_channels, hidden_channels)
        self.norm3 = LayerNorm(hidden_channels)
        self.conv4 = SAGEConv(hidden_channels, hidden_channels)
        self.norm4 = LayerNorm(hidden_channels)
        self.conv5 = SAGEConv(hidden_channels, hidden_channels)
        self.norm5 = LayerNorm(hidden_channels)

        self.lin = nn.Linear(hidden_channels, num_classes)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, edge_index, batch):

        x = self.conv1(x, edge_index)
        x = self.norm1(x)
        x = F.relu(x)
```

```python
        x_res = x
        x = self.conv2(x, edge_index)
        x = self.norm2(x)
        x = F.relu(x + x_res)

        x_res = x
        x = self.conv3(x, edge_index)
        x = self.norm3(x)
        x = F.relu(x + x_res)

        x_res = x
        x = self.conv4(x, edge_index)
        x = self.norm4(x)
        x = F.relu(x + x_res)

        x_res = x
        x = self.conv5(x, edge_index)
        x = self.norm5(x)
        x = F.relu(x + x_res)
        x = self.dropout(x)

        x = global_mean_pool(x, batch) #max

        x = self.lin(x)

        return x
```

In [7]:
```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GraphSAGE(
    in_channels=5,
    hidden_channels=256,
    num_classes=2,
    dropout=0.3
).to(device)

optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=5e-5)

base_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100, eta_min=5e-6)
scheduler = GradualWarmupScheduler(optimizer, multiplier=1.0, total_epoch=10, after_scheduler=base_scheduler)
```

In [8]:
```python
train_losses, val_losses, test_losses = [], [], []
train_accuracies, val_accuracies, test_accuracies = [], [], []
best_val_acc = 0
scaler = GradScaler()

def train():
    model.train()
    total_loss, correct, total = 0, 0, 0

    for data in train_loader:
        data = data.to(device)
        optimizer.zero_grad()

        with autocast():
            out = model(data.x, data.edge_index, data.batch)
            loss = F.cross_entropy(out, data.y.view(-1).long())

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        total_loss += loss.item()
        pred = out.argmax(dim=1)
        correct += (pred == data.y.view(-1)).sum().item()
        total += data.y.size(0)

    return total_loss / len(train_loader), correct / total

def evaluate(loader):
    model.eval()
    total_loss, correct, total = 0, 0, 0

    with torch.no_grad():
        for data in loader:
            data = data.to(device)
            out = model(data.x, data.edge_index, data.batch)
            loss = F.cross_entropy(out, data.y.view(-1).long())
            total_loss += loss.item()
            pred = out.argmax(dim=1)
            correct += (pred == data.y.view(-1)).sum().item()
            total += data.y.size(0)

    return total_loss / len(loader), correct / total
```

In [9]:
```python
num_epochs = 100 #80
for epoch in range(num_epochs):
```

```
    train_loss, train_acc = train()
    val_loss, val_acc = evaluate(val_loader)
    test_loss, test_acc = evaluate(test_loader)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    test_losses.append(test_loss)
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)
    test_accuracies.append(test_acc)

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(model.state_dict(), "best_model2.pth")

    scheduler.step()

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}, "
          f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}, Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.


model.load_state_dict(torch.load("best_model2.pth"))
final_test_loss, final_test_acc = evaluate(test_loader)
print(f"\n Final Test Accuracy: {final_test_acc:.4f}")

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label="Train Loss", color='red', linewidth=1.5)
plt.plot(range(1, num_epochs + 1), test_losses, label="Test Loss", color='blue', linewidth=1.5)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training, Test Loss")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label="Train Accuracy", color='red', linewidth=1.5)
plt.plot(range(1, num_epochs + 1), test_accuracies, label="Test Accuracy", color='blue', linewidth=1.5)
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training, Test Accuracy")
plt.legend()

plt.show()
```

```
Epoch 1/100, Train Loss: 0.8666, Train Acc: 0.5000, Val Loss: 0.8673, Val Acc: 0.5000, Test Loss: 0.8663, Test
Acc: 0.5000
Epoch 2/100, Train Loss: 0.6958, Train Acc: 0.5151, Val Loss: 0.6880, Val Acc: 0.5488, Test Loss: 0.6879, Test
Acc: 0.5534
Epoch 3/100, Train Loss: 0.6709, Train Acc: 0.5765, Val Loss: 0.6195, Val Acc: 0.6857, Test Loss: 0.6163, Test
Acc: 0.6840
Epoch 4/100, Train Loss: 0.6178, Train Acc: 0.6751, Val Loss: 0.6132, Val Acc: 0.6749, Test Loss: 0.6106, Test
Acc: 0.6757
Epoch 5/100, Train Loss: 0.6146, Train Acc: 0.6772, Val Loss: 0.6077, Val Acc: 0.6862, Test Loss: 0.6060, Test
Acc: 0.6843
Epoch 6/100, Train Loss: 0.6109, Train Acc: 0.6816, Val Loss: 0.6073, Val Acc: 0.6773, Test Loss: 0.6073, Test
Acc: 0.6767
Epoch 7/100, Train Loss: 0.6070, Train Acc: 0.6845, Val Loss: 0.6016, Val Acc: 0.6966, Test Loss: 0.6008, Test
Acc: 0.6936
Epoch 8/100, Train Loss: 0.6033, Train Acc: 0.6881, Val Loss: 0.5978, Val Acc: 0.6886, Test Loss: 0.5976, Test
Acc: 0.6880
Epoch 9/100, Train Loss: 0.5989, Train Acc: 0.6928, Val Loss: 0.5904, Val Acc: 0.7018, Test Loss: 0.5914, Test
Acc: 0.6978
Epoch 10/100, Train Loss: 0.5954, Train Acc: 0.6961, Val Loss: 0.5868, Val Acc: 0.7050, Test Loss: 0.5884, Test
Acc: 0.7011
```

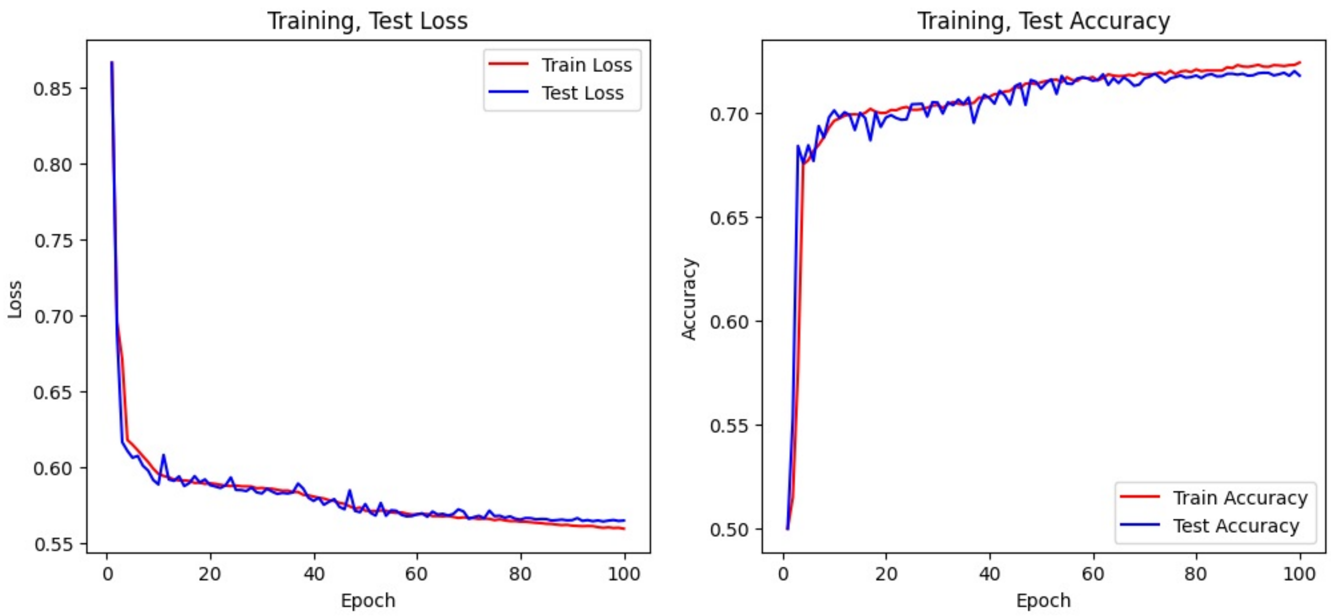
```
/beegfs/home/anning/.conda/envs/qenv/lib/python3.10/site-packages/torch/optim/lr_scheduler.py:855: UserWarning:
To get the last learning rate computed by the scheduler, please use `get_last_lr()`.
  warnings.warn("To get the last learning rate computed by the scheduler, "
```

```
Epoch 11/100, Train Loss: 0.5940, Train Acc: 0.6972, Val Loss: 0.6073, Val Acc: 0.7018, Test Loss: 0.6079, Test
Acc: 0.6975
Epoch 12/100, Train Loss: 0.5932, Train Acc: 0.6985, Val Loss: 0.5882, Val Acc: 0.7057, Test Loss: 0.5917, Test
Acc: 0.7002
Epoch 13/100, Train Loss: 0.5916, Train Acc: 0.6991, Val Loss: 0.5878, Val Acc: 0.7055, Test Loss: 0.5908, Test
Acc: 0.6988
Epoch 14/100, Train Loss: 0.5911, Train Acc: 0.6992, Val Loss: 0.5908, Val Acc: 0.6966, Test Loss: 0.5938, Test
Acc: 0.6917
Epoch 15/100, Train Loss: 0.5911, Train Acc: 0.6989, Val Loss: 0.5868, Val Acc: 0.7003, Test Loss: 0.5873, Test
Acc: 0.6999
Epoch 16/100, Train Loss: 0.5907, Train Acc: 0.6999, Val Loss: 0.5893, Val Acc: 0.7003, Test Loss: 0.5894, Test
Acc: 0.6974
Epoch 17/100, Train Loss: 0.5894, Train Acc: 0.7018, Val Loss: 0.5922, Val Acc: 0.6860, Test Loss: 0.5939, Test
Acc: 0.6867
Epoch 18/100, Train Loss: 0.5896, Train Acc: 0.7005, Val Loss: 0.5886, Val Acc: 0.7061, Test Loss: 0.5898, Test
Acc: 0.7002
Epoch 19/100, Train Loss: 0.5888, Train Acc: 0.6998, Val Loss: 0.5914, Val Acc: 0.6931, Test Loss: 0.5918, Test
Acc: 0.6931
Epoch 20/100, Train Loss: 0.5892, Train Acc: 0.6999, Val Loss: 0.5866, Val Acc: 0.6999, Test Loss: 0.5880, Test
Acc: 0.6977
Epoch 21/100, Train Loss: 0.5888, Train Acc: 0.7013, Val Loss: 0.5862, Val Acc: 0.7034, Test Loss: 0.5871, Test
```

```
Acc: 0.6988
Epoch 22/100, Train Loss: 0.5882, Train Acc: 0.7012, Val Loss: 0.5831, Val Acc: 0.7023, Test Loss: 0.5861, Test
Acc: 0.6974
Epoch 23/100, Train Loss: 0.5876, Train Acc: 0.7022, Val Loss: 0.5868, Val Acc: 0.7006, Test Loss: 0.5879, Test
Acc: 0.6966
Epoch 24/100, Train Loss: 0.5873, Train Acc: 0.7027, Val Loss: 0.5877, Val Acc: 0.7033, Test Loss: 0.5931, Test
Acc: 0.6968
Epoch 25/100, Train Loss: 0.5877, Train Acc: 0.7014, Val Loss: 0.5825, Val Acc: 0.7075, Test Loss: 0.5848, Test
Acc: 0.7040
Epoch 26/100, Train Loss: 0.5872, Train Acc: 0.7014, Val Loss: 0.5824, Val Acc: 0.7081, Test Loss: 0.5847, Test
Acc: 0.7041
Epoch 27/100, Train Loss: 0.5871, Train Acc: 0.7017, Val Loss: 0.5824, Val Acc: 0.7094, Test Loss: 0.5840, Test
Acc: 0.7043
Epoch 28/100, Train Loss: 0.5870, Train Acc: 0.7024, Val Loss: 0.5845, Val Acc: 0.6996, Test Loss: 0.5864, Test
Acc: 0.6981
Epoch 29/100, Train Loss: 0.5859, Train Acc: 0.7033, Val Loss: 0.5802, Val Acc: 0.7098, Test Loss: 0.5832, Test
Acc: 0.7050
Epoch 30/100, Train Loss: 0.5861, Train Acc: 0.7036, Val Loss: 0.5802, Val Acc: 0.7094, Test Loss: 0.5825, Test
Acc: 0.7049
Epoch 31/100, Train Loss: 0.5857, Train Acc: 0.7023, Val Loss: 0.5822, Val Acc: 0.7031, Test Loss: 0.5856, Test
Acc: 0.6996
Epoch 32/100, Train Loss: 0.5856, Train Acc: 0.7040, Val Loss: 0.5810, Val Acc: 0.7106, Test Loss: 0.5837, Test
Acc: 0.7050
Epoch 33/100, Train Loss: 0.5849, Train Acc: 0.7052, Val Loss: 0.5788, Val Acc: 0.7097, Test Loss: 0.5821, Test
Acc: 0.7036
Epoch 34/100, Train Loss: 0.5843, Train Acc: 0.7043, Val Loss: 0.5784, Val Acc: 0.7122, Test Loss: 0.5829, Test
Acc: 0.7063
Epoch 35/100, Train Loss: 0.5844, Train Acc: 0.7039, Val Loss: 0.5802, Val Acc: 0.7095, Test Loss: 0.5824, Test
Acc: 0.7041
Epoch 36/100, Train Loss: 0.5835, Train Acc: 0.7045, Val Loss: 0.5800, Val Acc: 0.7115, Test Loss: 0.5832, Test
Acc: 0.7073
Epoch 37/100, Train Loss: 0.5834, Train Acc: 0.7047, Val Loss: 0.5863, Val Acc: 0.6974, Test Loss: 0.5890, Test
Acc: 0.6951
Epoch 38/100, Train Loss: 0.5816, Train Acc: 0.7073, Val Loss: 0.5815, Val Acc: 0.7103, Test Loss: 0.5857, Test
Acc: 0.7033
Epoch 39/100, Train Loss: 0.5813, Train Acc: 0.7071, Val Loss: 0.5761, Val Acc: 0.7146, Test Loss: 0.5797, Test
Acc: 0.7086
Epoch 40/100, Train Loss: 0.5804, Train Acc: 0.7076, Val Loss: 0.5756, Val Acc: 0.7145, Test Loss: 0.5776, Test
Acc: 0.7071
Epoch 41/100, Train Loss: 0.5797, Train Acc: 0.7089, Val Loss: 0.5777, Val Acc: 0.7079, Test Loss: 0.5796, Test
Acc: 0.7043
Epoch 42/100, Train Loss: 0.5791, Train Acc: 0.7091, Val Loss: 0.5704, Val Acc: 0.7163, Test Loss: 0.5750, Test
Acc: 0.7105
Epoch 43/100, Train Loss: 0.5780, Train Acc: 0.7102, Val Loss: 0.5743, Val Acc: 0.7131, Test Loss: 0.5770, Test
Acc: 0.7082
Epoch 44/100, Train Loss: 0.5775, Train Acc: 0.7104, Val Loss: 0.5755, Val Acc: 0.7091, Test Loss: 0.5788, Test
Acc: 0.7039
Epoch 45/100, Train Loss: 0.5763, Train Acc: 0.7125, Val Loss: 0.5704, Val Acc: 0.7158, Test Loss: 0.5736, Test
Acc: 0.7128
Epoch 46/100, Train Loss: 0.5756, Train Acc: 0.7118, Val Loss: 0.5697, Val Acc: 0.7168, Test Loss: 0.5720, Test
Acc: 0.7139
Epoch 47/100, Train Loss: 0.5740, Train Acc: 0.7139, Val Loss: 0.5834, Val Acc: 0.7075, Test Loss: 0.5846, Test
Acc: 0.7037
Epoch 48/100, Train Loss: 0.5725, Train Acc: 0.7141, Val Loss: 0.5679, Val Acc: 0.7176, Test Loss: 0.5710, Test
Acc: 0.7157
Epoch 49/100, Train Loss: 0.5732, Train Acc: 0.7137, Val Loss: 0.5676, Val Acc: 0.7170, Test Loss: 0.5701, Test
Acc: 0.7148
Epoch 50/100, Train Loss: 0.5711, Train Acc: 0.7146, Val Loss: 0.5733, Val Acc: 0.7144, Test Loss: 0.5755, Test
Acc: 0.7115
Epoch 51/100, Train Loss: 0.5709, Train Acc: 0.7152, Val Loss: 0.5662, Val Acc: 0.7174, Test Loss: 0.5697, Test
Acc: 0.7142
Epoch 52/100, Train Loss: 0.5709, Train Acc: 0.7157, Val Loss: 0.5655, Val Acc: 0.7189, Test Loss: 0.5679, Test
Acc: 0.7160
Epoch 53/100, Train Loss: 0.5712, Train Acc: 0.7159, Val Loss: 0.5731, Val Acc: 0.7116, Test Loss: 0.5762, Test
Acc: 0.7090
Epoch 54/100, Train Loss: 0.5711, Train Acc: 0.7147, Val Loss: 0.5655, Val Acc: 0.7193, Test Loss: 0.5678, Test
Acc: 0.7177
Epoch 55/100, Train Loss: 0.5700, Train Acc: 0.7171, Val Loss: 0.5693, Val Acc: 0.7151, Test Loss: 0.5714, Test
Acc: 0.7140
Epoch 56/100, Train Loss: 0.5697, Train Acc: 0.7157, Val Loss: 0.5683, Val Acc: 0.7180, Test Loss: 0.5710, Test
Acc: 0.7138
Epoch 57/100, Train Loss: 0.5700, Train Acc: 0.7153, Val Loss: 0.5670, Val Acc: 0.7169, Test Loss: 0.5684, Test
Acc: 0.7164
Epoch 58/100, Train Loss: 0.5691, Train Acc: 0.7170, Val Loss: 0.5653, Val Acc: 0.7172, Test Loss: 0.5674, Test
Acc: 0.7169
Epoch 59/100, Train Loss: 0.5686, Train Acc: 0.7165, Val Loss: 0.5653, Val Acc: 0.7170, Test Loss: 0.5676, Test
Acc: 0.7165
Epoch 60/100, Train Loss: 0.5687, Train Acc: 0.7171, Val Loss: 0.5676, Val Acc: 0.7155, Test Loss: 0.5687, Test
Acc: 0.7153
Epoch 61/100, Train Loss: 0.5689, Train Acc: 0.7155, Val Loss: 0.5654, Val Acc: 0.7173, Test Loss: 0.5694, Test
Acc: 0.7158
Epoch 62/100, Train Loss: 0.5689, Train Acc: 0.7167, Val Loss: 0.5661, Val Acc: 0.7185, Test Loss: 0.5671, Test
Acc: 0.7184
Epoch 63/100, Train Loss: 0.5675, Train Acc: 0.7184, Val Loss: 0.5665, Val Acc: 0.7165, Test Loss: 0.5705, Test
Acc: 0.7133
Epoch 64/100, Train Loss: 0.5676, Train Acc: 0.7178, Val Loss: 0.5675, Val Acc: 0.7142, Test Loss: 0.5686, Test
Acc: 0.7166
Epoch 65/100, Train Loss: 0.5673, Train Acc: 0.7175, Val Loss: 0.5652, Val Acc: 0.7170, Test Loss: 0.5690, Test
Acc: 0.7142
```

```
Epoch 66/100, Train Loss: 0.5677, Train Acc: 0.7176, Val Loss: 0.5660, Val Acc: 0.7157, Test Loss: 0.5679, Test Acc: 0.7170
Epoch 67/100, Train Loss: 0.5670, Train Acc: 0.7182, Val Loss: 0.5647, Val Acc: 0.7170, Test Loss: 0.5689, Test Acc: 0.7153
Epoch 68/100, Train Loss: 0.5663, Train Acc: 0.7177, Val Loss: 0.5694, Val Acc: 0.7152, Test Loss: 0.5720, Test Acc: 0.7129
Epoch 69/100, Train Loss: 0.5667, Train Acc: 0.7190, Val Loss: 0.5680, Val Acc: 0.7158, Test Loss: 0.5706, Test Acc: 0.7135
Epoch 70/100, Train Loss: 0.5664, Train Acc: 0.7183, Val Loss: 0.5638, Val Acc: 0.7175, Test Loss: 0.5657, Test Acc: 0.7165
Epoch 71/100, Train Loss: 0.5661, Train Acc: 0.7185, Val Loss: 0.5649, Val Acc: 0.7185, Test Loss: 0.5668, Test Acc: 0.7173
Epoch 72/100, Train Loss: 0.5657, Train Acc: 0.7185, Val Loss: 0.5639, Val Acc: 0.7209, Test Loss: 0.5677, Test Acc: 0.7187
Epoch 73/100, Train Loss: 0.5659, Train Acc: 0.7193, Val Loss: 0.5638, Val Acc: 0.7192, Test Loss: 0.5661, Test Acc: 0.7169
Epoch 74/100, Train Loss: 0.5657, Train Acc: 0.7185, Val Loss: 0.5699, Val Acc: 0.7180, Test Loss: 0.5711, Test Acc: 0.7146
Epoch 75/100, Train Loss: 0.5649, Train Acc: 0.7201, Val Loss: 0.5646, Val Acc: 0.7198, Test Loss: 0.5675, Test Acc: 0.7165
Epoch 76/100, Train Loss: 0.5655, Train Acc: 0.7185, Val Loss: 0.5644, Val Acc: 0.7185, Test Loss: 0.5678, Test Acc: 0.7171
Epoch 77/100, Train Loss: 0.5647, Train Acc: 0.7198, Val Loss: 0.5623, Val Acc: 0.7201, Test Loss: 0.5665, Test Acc: 0.7179
Epoch 78/100, Train Loss: 0.5642, Train Acc: 0.7202, Val Loss: 0.5639, Val Acc: 0.7190, Test Loss: 0.5674, Test Acc: 0.7168
Epoch 79/100, Train Loss: 0.5643, Train Acc: 0.7195, Val Loss: 0.5623, Val Acc: 0.7184, Test Loss: 0.5658, Test Acc: 0.7171
Epoch 80/100, Train Loss: 0.5638, Train Acc: 0.7208, Val Loss: 0.5620, Val Acc: 0.7196, Test Loss: 0.5653, Test Acc: 0.7179
Epoch 81/100, Train Loss: 0.5638, Train Acc: 0.7199, Val Loss: 0.5625, Val Acc: 0.7188, Test Loss: 0.5663, Test Acc: 0.7166
Epoch 82/100, Train Loss: 0.5635, Train Acc: 0.7203, Val Loss: 0.5626, Val Acc: 0.7198, Test Loss: 0.5662, Test Acc: 0.7181
Epoch 83/100, Train Loss: 0.5631, Train Acc: 0.7202, Val Loss: 0.5616, Val Acc: 0.7196, Test Loss: 0.5654, Test Acc: 0.7185
Epoch 84/100, Train Loss: 0.5629, Train Acc: 0.7202, Val Loss: 0.5623, Val Acc: 0.7199, Test Loss: 0.5657, Test Acc: 0.7174
Epoch 85/100, Train Loss: 0.5625, Train Acc: 0.7203, Val Loss: 0.5621, Val Acc: 0.7203, Test Loss: 0.5656, Test Acc: 0.7175
Epoch 86/100, Train Loss: 0.5624, Train Acc: 0.7218, Val Loss: 0.5607, Val Acc: 0.7200, Test Loss: 0.5646, Test Acc: 0.7187
Epoch 87/100, Train Loss: 0.5619, Train Acc: 0.7214, Val Loss: 0.5603, Val Acc: 0.7197, Test Loss: 0.5649, Test Acc: 0.7187
Epoch 88/100, Train Loss: 0.5616, Train Acc: 0.7228, Val Loss: 0.5608, Val Acc: 0.7201, Test Loss: 0.5652, Test Acc: 0.7183
Epoch 89/100, Train Loss: 0.5618, Train Acc: 0.7222, Val Loss: 0.5605, Val Acc: 0.7203, Test Loss: 0.5648, Test Acc: 0.7187
Epoch 90/100, Train Loss: 0.5612, Train Acc: 0.7220, Val Loss: 0.5608, Val Acc: 0.7196, Test Loss: 0.5649, Test Acc: 0.7178
Epoch 91/100, Train Loss: 0.5610, Train Acc: 0.7224, Val Loss: 0.5624, Val Acc: 0.7185, Test Loss: 0.5662, Test Acc: 0.7179
Epoch 92/100, Train Loss: 0.5609, Train Acc: 0.7230, Val Loss: 0.5600, Val Acc: 0.7199, Test Loss: 0.5644, Test Acc: 0.7189
Epoch 93/100, Train Loss: 0.5610, Train Acc: 0.7221, Val Loss: 0.5605, Val Acc: 0.7205, Test Loss: 0.5649, Test Acc: 0.7192
Epoch 94/100, Train Loss: 0.5609, Train Acc: 0.7220, Val Loss: 0.5603, Val Acc: 0.7213, Test Loss: 0.5642, Test Acc: 0.7191
Epoch 95/100, Train Loss: 0.5603, Train Acc: 0.7228, Val Loss: 0.5601, Val Acc: 0.7205, Test Loss: 0.5647, Test Acc: 0.7179
Epoch 96/100, Train Loss: 0.5598, Train Acc: 0.7227, Val Loss: 0.5602, Val Acc: 0.7209, Test Loss: 0.5640, Test Acc: 0.7184
Epoch 97/100, Train Loss: 0.5602, Train Acc: 0.7224, Val Loss: 0.5602, Val Acc: 0.7199, Test Loss: 0.5646, Test Acc: 0.7192
Epoch 98/100, Train Loss: 0.5597, Train Acc: 0.7228, Val Loss: 0.5603, Val Acc: 0.7195, Test Loss: 0.5650, Test Acc: 0.7177
Epoch 99/100, Train Loss: 0.5599, Train Acc: 0.7229, Val Loss: 0.5596, Val Acc: 0.7221, Test Loss: 0.5643, Test Acc: 0.7200
Epoch 100/100, Train Loss: 0.5592, Train Acc: 0.7241, Val Loss: 0.5601, Val Acc: 0.7201, Test Loss: 0.5646, Test Acc: 0.7178

 Final Test Accuracy: 0.7200
```


Training, Test Loss
Train Loss
Test Loss
Loss
Epoch
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
20
40
60
80
100
Training, Test Accuracy
Train Accuracy
Test Accuracy
Accuracy
Epoch
0.70
0.65
0.60
0.55
0.50