Question 3

```
# @title Install & Imports
!pip -q install torch torchvision torchaudio --index-url https://download.pyto
!pip -q install scikit-learn matplotlib tqdm >/dev/null

import os, time, random, math
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Subset, TensorDataset
from torchvision import datasets, transforms, models
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, Confus
from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from tqdm.auto import tqdm

# Reproducibility
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print("Device:", DEVICE)

# Output directory
OUT_DIR = "/content/outputs"
os.makedirs(OUT_DIR, exist_ok=True)

# Small helpers
def count_params(model, only_trainable=False):
    params = [p.numel() for p in model.parameters() if (p.requires_grad or no
    return sum(params)

def time_inference(model, loader, device=DEVICE):
    model.eval()
    torch.cuda.synchronize() if device=="cuda" else None
    t0 = time.time()
    with torch.no_grad():
        for xb, yb in loader:
            xb = xb.to(device)
            _ = model(xb)
```

```
    torch.cuda.synchronize() if device=="cuda" else None
    return time.time()-t0

def plot_confmat(y_true, y_pred, title, fname):
    cm = confusion_matrix(y_true, y_pred, labels=list(range(10)))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=list(ran
    fig, ax = plt.subplots(figsize=(6,6))
    disp.plot(ax=ax, colorbar=False)
    ax.set_title(title)
    plt.tight_layout()
    fig.savefig(fname, dpi=150)
    plt.show()

def tsne_plot(z, labels, title, fname):
    z2 = TSNE(n_components=2, init="pca", random_state=SEED, perplexity=30).fi
    plt.figure(figsize=(6,5))
    scatter = plt.scatter(z2[:,0], z2[:,1], c=labels, s=6, alpha=0.8)
    plt.title(title)
    plt.xticks([]); plt.yticks([])
    plt.tight_layout()
    plt.savefig(fname, dpi=150)
    plt.show()

Device: cpu

# @title Download MNIST & Fashion-MNIST
BATCH = 128

tfm_basic = transforms.Compose([
    transforms.ToTensor(),                # [0,1]
    transforms.Normalize((0.1307,), (0.3081,))  # standard MNIST stats
])

mnist_train = datasets.MNIST(root="/content", train=True, download=True, trans
mnist_test  = datasets.MNIST(root="/content", train=False, download=True, tran

fmnist_test = datasets.FashionMNIST(root="/content", train=False, download=Tru

# Optional: stratified subset of training to speed up (set to None to use full
SUBSET_PER_CLASS = 1000  # e.g., 1000 => 10k total. Set to None to use full.
if SUBSET_PER_CLASS is not None:
    targets = np.array(mnist_train.targets)
    idxs = []
    for c in range(10):
        c_idx = np.where(targets==c)[0]
        np.random.shuffle(c_idx)
        idxs.extend(c_idx[:SUBSET_PER_CLASS])
    mnist_train = Subset(mnist_train, idxs)
    print(f"Using stratified subset: {len(mnist_train)} samples")
```

```
else:
    print("Using full training set")

train_loader = DataLoader(mnist_train, batch_size=BATCH, shuffle=True, num_wo
test_loader  = DataLoader(mnist_test,  batch_size=BATCH, shuffle=False, num_wo

# Also create flat versions (for sklearn baselines)
def dataset_to_numpy(dset):
    xs, ys = [], []
    for x, y in DataLoader(dset, batch_size=512, shuffle=False):
        xs.append(x.numpy())
        ys.append(y.numpy())
    X = np.concatenate(xs, axis=0) # N,1,28,28
    y = np.concatenate(ys, axis=0)
    return X, y

Xtr, ytr = dataset_to_numpy(mnist_train)
Xte, yte = dataset_to_numpy(mnist_test)
Xtr_flat = Xtr.reshape(len(Xtr), -1)
Xte_flat = Xte.reshape(len(Xte), -1)
print("Train flat:", Xtr_flat.shape, "Test flat:", Xte_flat.shape)

# Fashion-MNIST test in numpy (for cross-domain eval with MLP embeddings)
Xf_te, yf_te = dataset_to_numpy(fmnist_test)
```

```
Using stratified subset: 10000 samples
Train flat: (10000, 784) Test flat: (10000, 784)
```

3.1

```
# @title Define MLP (30→20→10) and trainer
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28*28, 30)
        self.fc2 = nn.Linear(30, 20)
        self.fc3 = nn.Linear(20, 10)
    def features20(self, x):  # returns the 20-dim hidden layer
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)  # no ReLU here so we can see raw embeddings
        return x
    def forward(self, x):
        z = self.features20(x)
        x = F.relu(z)
        x = self.fc3(x)
        return x
```

```
def train_classifier(model, loader, epochs=5, lr=1e-3, wd=0.0, device=DEVICE)
    model.to(device)
    opt = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=wd)
    criterion = nn.CrossEntropyLoss()
    model.train()
    for ep in range(epochs):
        pbar = tqdm(loader, leave=False)
        running = 0.0
        for xb, yb in pbar:
            xb, yb = xb.to(device), yb.to(device)
            opt.zero_grad()
            logits = model(xb)
            loss = criterion(logits, yb)
            loss.backward()
            opt.step()
            running += loss.item()*xb.size(0)
            pbar.set_description(f"Epoch {ep+1}/{epochs} loss={running/((pbar
    return model

def predict(model, loader, device=DEVICE):
    model.eval()
    preds, golds = [], []
    with torch.no_grad():
        for xb, yb in loader:
            logits = model(xb.to(device))
            pred = logits.argmax(dim=1).cpu().numpy()
            preds.append(pred)
            golds.append(yb.numpy())
    return np.concatenate(preds), np.concatenate(golds)

# @title Train MLP and evaluate against RandomForest & LogisticRegression
# MLP
mlp = MLP()
mlp = train_classifier(mlp, train_loader, epochs=10, lr=1e-3, wd=1e-4)

y_pred_mlp, y_true = predict(mlp, test_loader)
acc_mlp = accuracy_score(y_true, y_pred_mlp)
f1_mlp  = f1_score(y_true, y_pred_mlp, average="macro")
print(f"MLP — Acc: {acc_mlp:.4f}, F1(macro): {f1_mlp:.4f}")
plot_confmat(y_true, y_pred_mlp, "MLP Confusion Matrix (MNIST test)", f"{OUT_

# Random Forest (sklearn)
rf = RandomForestClassifier(n_estimators=200, max_depth=None, random_state=SE
rf.fit(Xtr_flat, ytr)
y_pred_rf = rf.predict(Xte_flat)
acc_rf = accuracy_score(yte, y_pred_rf)
f1_rf  = f1_score(yte, y_pred_rf, average="macro")
print(f"RandomForest — Acc: {acc_rf:.4f}, F1(macro): {f1_rf:.4f}")
```

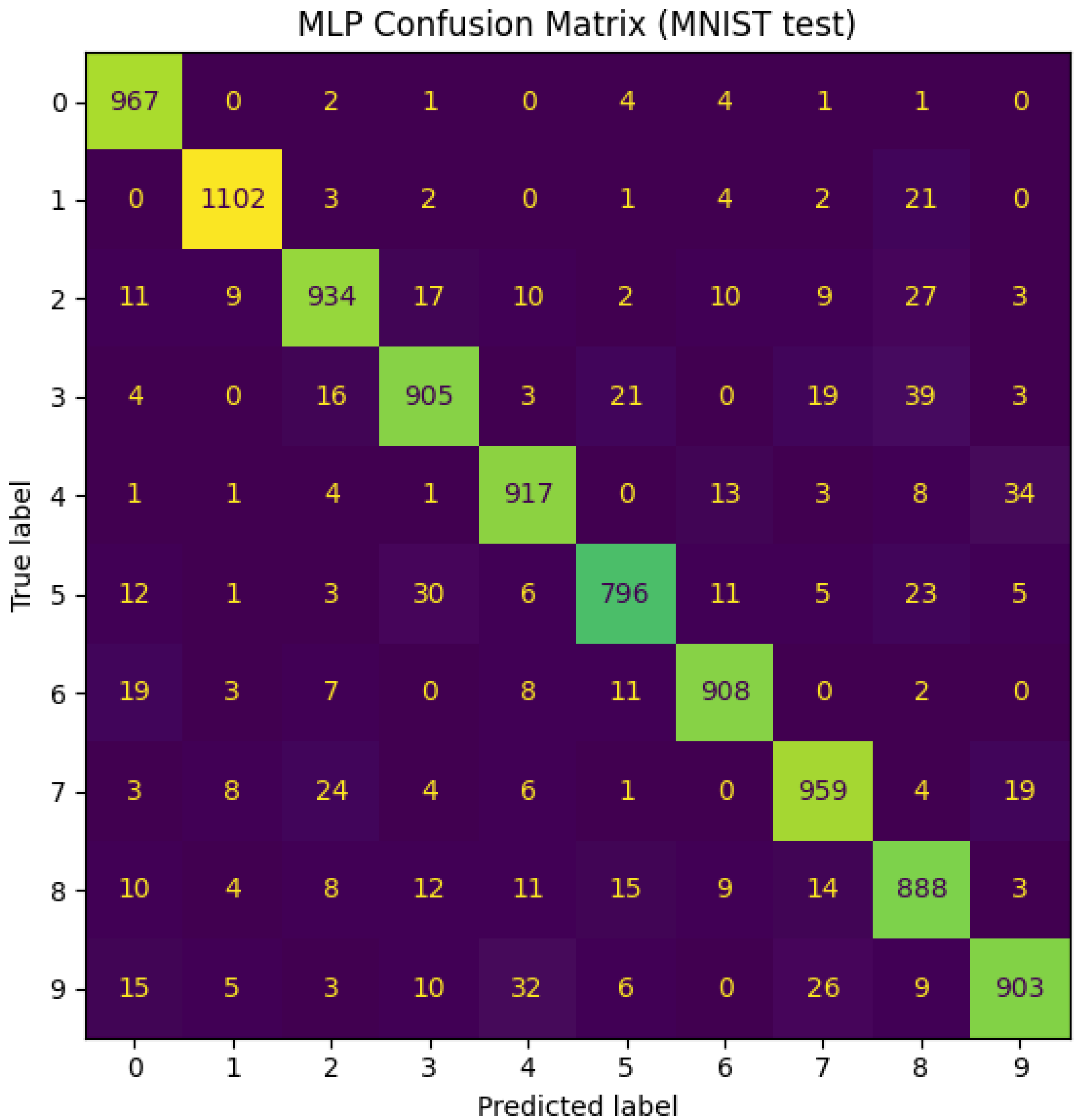
```
plot_confmat(yte, y_pred_rf, "RandomForest Confusion Matrix (MNIST test)", f"

# Logistic Regression (sklearn) – multinomial saga is fast & good
lr = LogisticRegression(max_iter=200, solver="saga", penalty="l2", n_jobs=-1,
lr.fit(Xtr_flat, ytr)
y_pred_lr = lr.predict(Xte_flat)
acc_lr = accuracy_score(yte, y_pred_lr)
f1_lr  = f1_score(yte, y_pred_lr, average="macro")
print(f"LogReg – Acc: {acc_lr:.4f}, F1(macro): {f1_lr:.4f}")
plot_confmat(yte, y_pred_lr, "Logistic Regression Confusion Matrix (MNIST tes
```

```
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:666: Us
  warnings.warn(warn_msg)



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]


Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x78
<function _MultiProcessingDataLoaderIter.__del__ at 0x78050cc12020>Traceback

  File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py
Traceback (most recent call last):
      File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloade
self._shutdown_workers()
self._shutdown_workers()  File "/usr/local/lib/python3.12/dist-packages/torch/

      File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloade
if w.is_alive():
 if w.is_alive():
       ^ ^ ^^^ ^^^^^^^^^^^^^^^
^  File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_aliv
```

```
^    assert self._parent_pid == os.getpid(), 'can only test a child process'^
 ^
    File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_ali
      assert self._parent_pid == os.getpid(), 'can only test a child process'
      ^^ ^^ ^ ^^ ^  ^ ^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^AssertionError^: ^^can only test a child process^
^Exception ignored in: ^^<function _MultiProcessingDataLoaderIter.__del__ at 0
Traceback (most recent call last):
^
AssertionError  File "/usr/local/lib/python3.12/dist-packages/torch/utils/data
: can only test a child process    self._shutdown_workers()

Exception ignored in:   File "/usr/local/lib/python3.12/dist-packages/torch/ut
<function _MultiProcessingDataLoaderIter.__del__ at 0x78050cc12020>
if w.is_alive():Traceback (most recent call last):

   File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.p
      self._shutdown_workers()
   File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.p
      ^if w.is_alive():^
 ^^ ^  ^ ^  ^^^^^^^^^^^
  File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_alive
    ^assert self._parent_pid == os.getpid(), 'can only test a child process'^
^ ^ ^  ^
    File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_ali
       assert self._parent_pid == os.getpid(), 'can only test a child process
  ^ ^ ^^  ^  ^ ^ ^ ^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^AssertionError: can only test a child process^
^^Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x
^Traceback (most recent call last):
^^  File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader
^    ^self._shutdown_workers()^

  File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py
AssertionError    : can only test a child process
if w.is_alive():Exception ignored in:
 <function _MultiProcessingDataLoaderIter.__del__ at 0x78050cc12020>
Traceback (most recent call last):
   File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.p
      self._shutdown_workers() ^
^  File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.p
^    if w.is_alive():^
^ ^  ^ ^ ^ ^ ^^^^
^  File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_aliv
^    ^assert self._parent_pid == os.getpid(), 'can only test a child process'
^^^  ^  ^ ^ ^
   File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_aliv
     assert self._parent_pid == os.getpid(), 'can only test a child process'
```

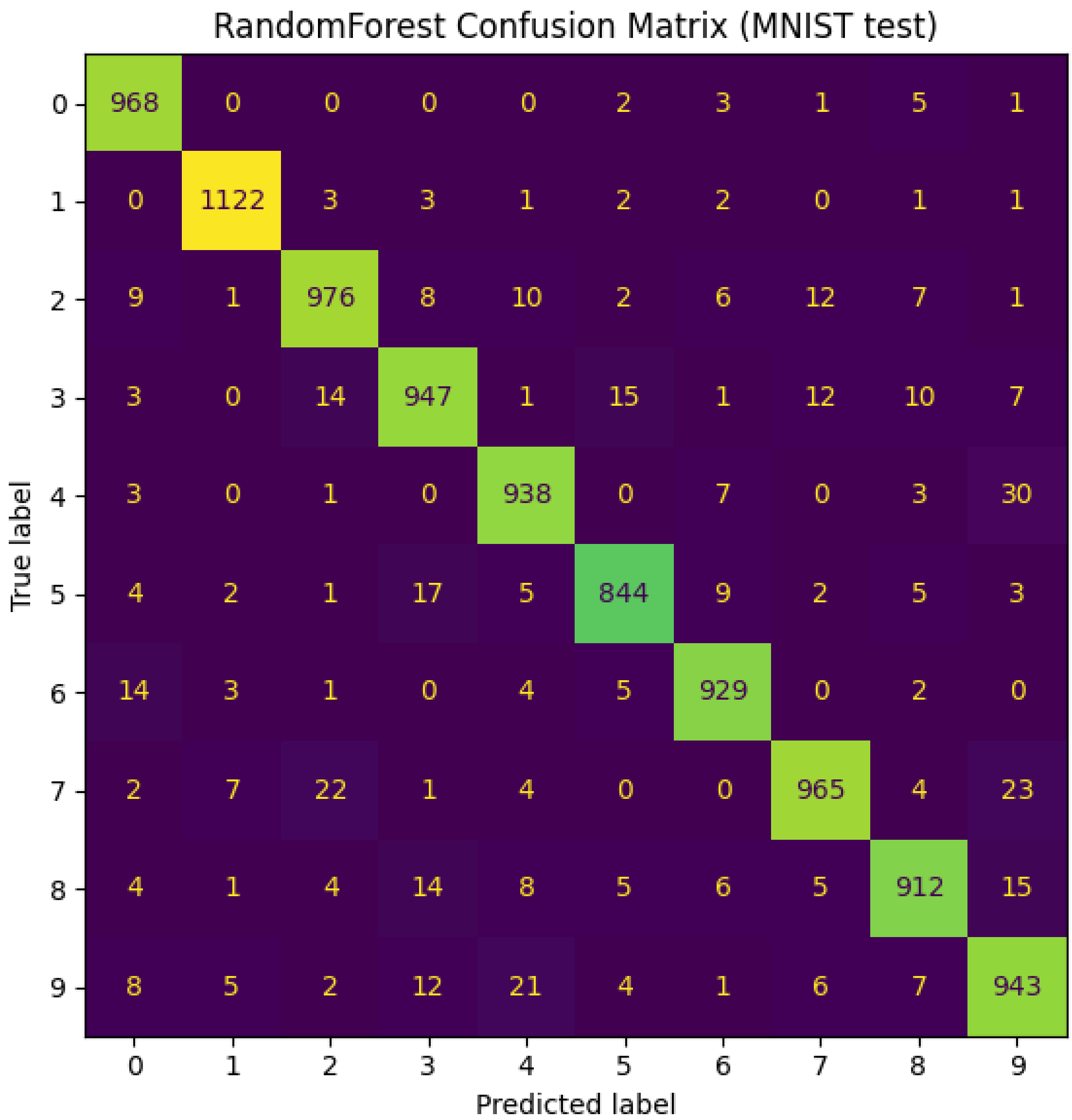
```
 ^ ^^^ ^ ^ ^ ^ ^^  ^  ^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^AssertionError: ^can only test a child process
^Exception ignored in: ^<function _MultiProcessingDataLoaderIter.__del__ at 0x
^Traceback (most recent call last):
^^^  File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloade
^
    AssertionErrorself._shutdown_workers():
can only test a child process  File "/usr/local/lib/python3.12/dist-packages/t

    if w.is_alive():Exception ignored in: <function _MultiProcessingDataLoade

Traceback (most recent call last):
   File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.p
      self._shutdown_workers()
   File "/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.p
     if w.is_alive():
   ^ ^^ ^ ^ ^ ^^^^^^^^^^^^^^^
^  File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_aliv
    ^^assert self._parent_pid == os.getpid(), 'can only test a child process'

   File "/usr/lib/python3.12/multiprocessing/process.py", line 160, in is_aliv
        assert self._parent_pid == os.getpid(), 'can only test a child process
             ^  ^ ^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^AssertionError^: ^can only test a child process^
^
AssertionError: can only test a child process



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]



  0%|          | 0/79 [00:00<?, ?it/s]
```

MLP — Acc: 0.9279, F1(macro): 0.9271



png

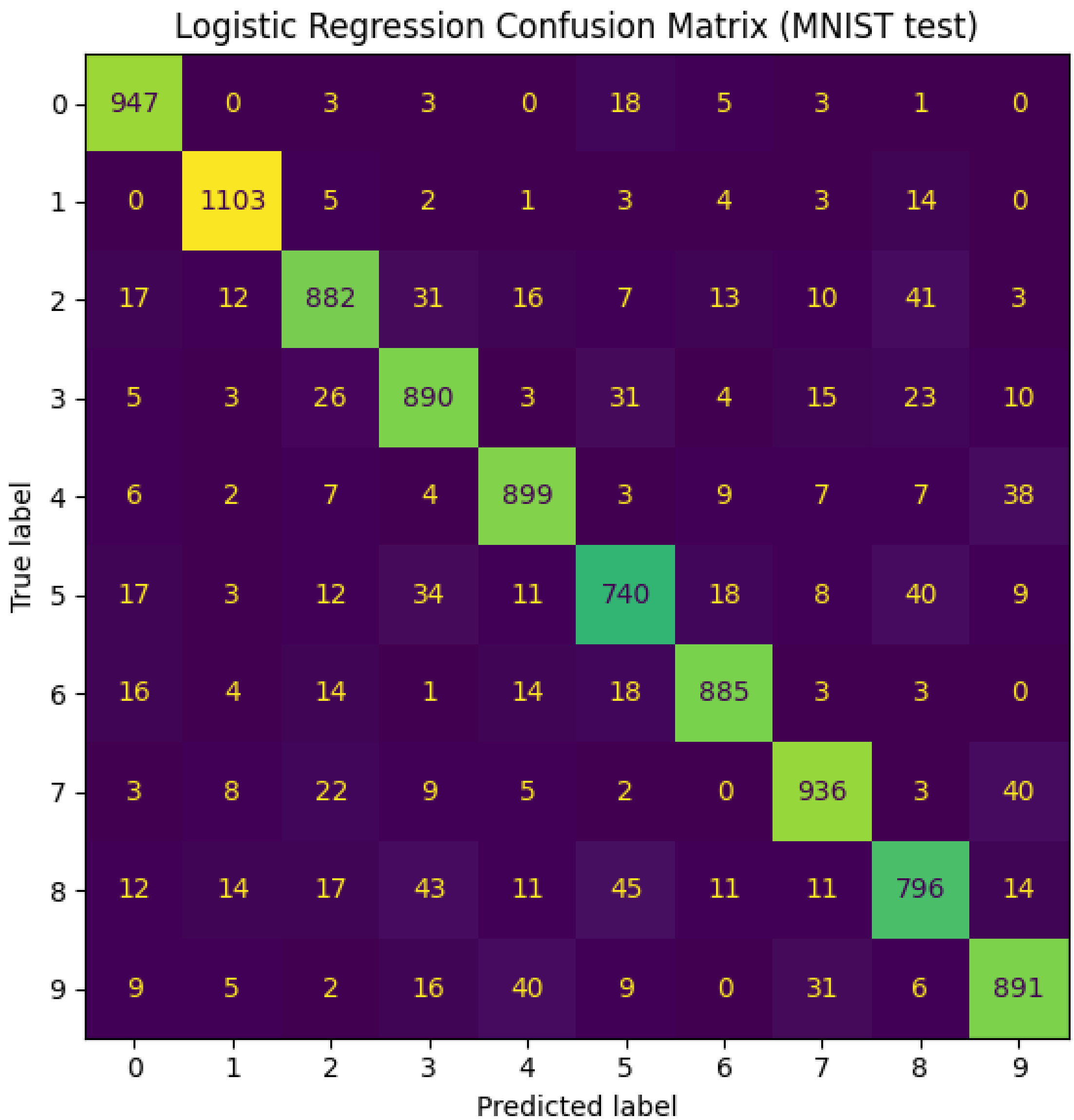
RandomForest — Acc: 0.9544, F1(macro): 0.9540



png

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_sag.py:348: Conv
  warnings.warn(
```

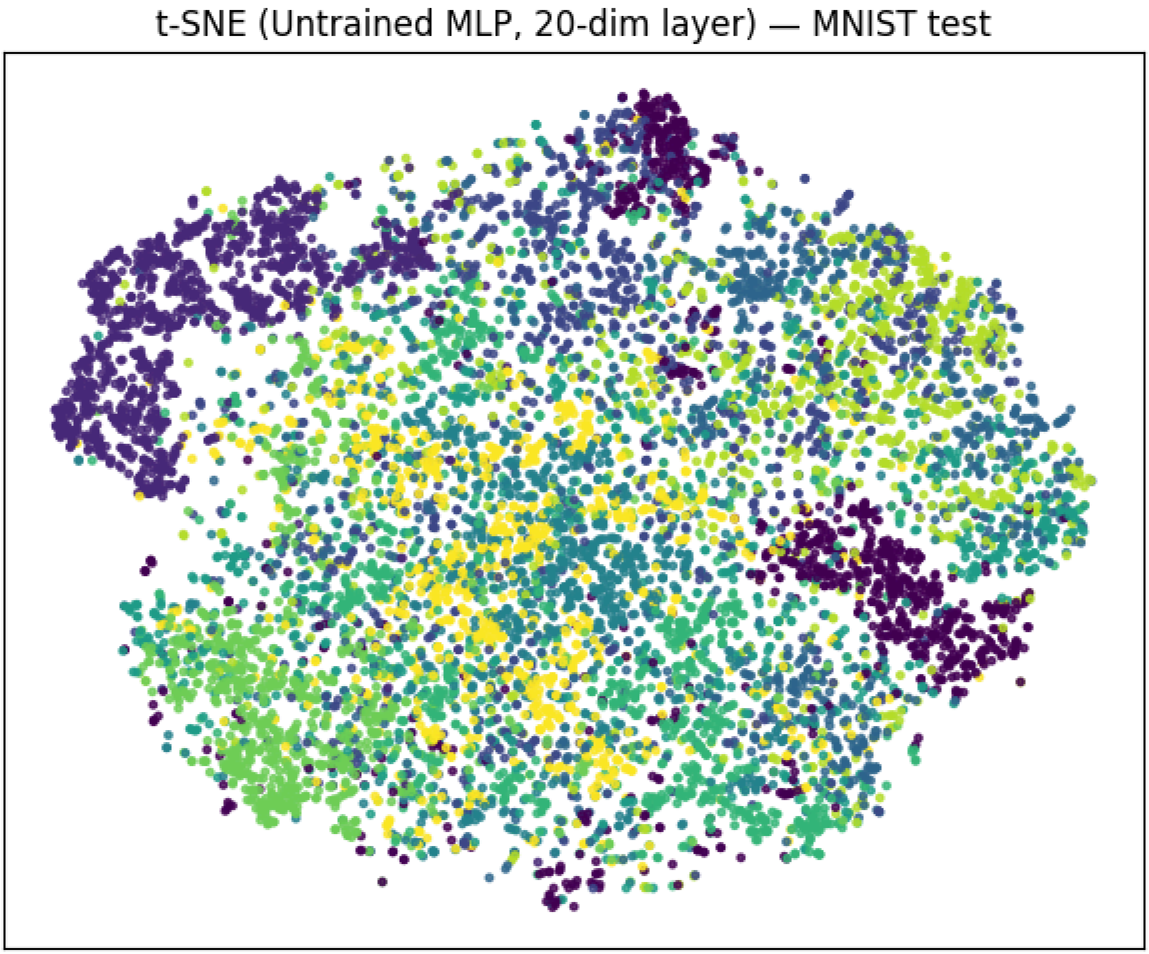
```
LogReg — Acc: 0.8969, F1(macro): 0.8951
```



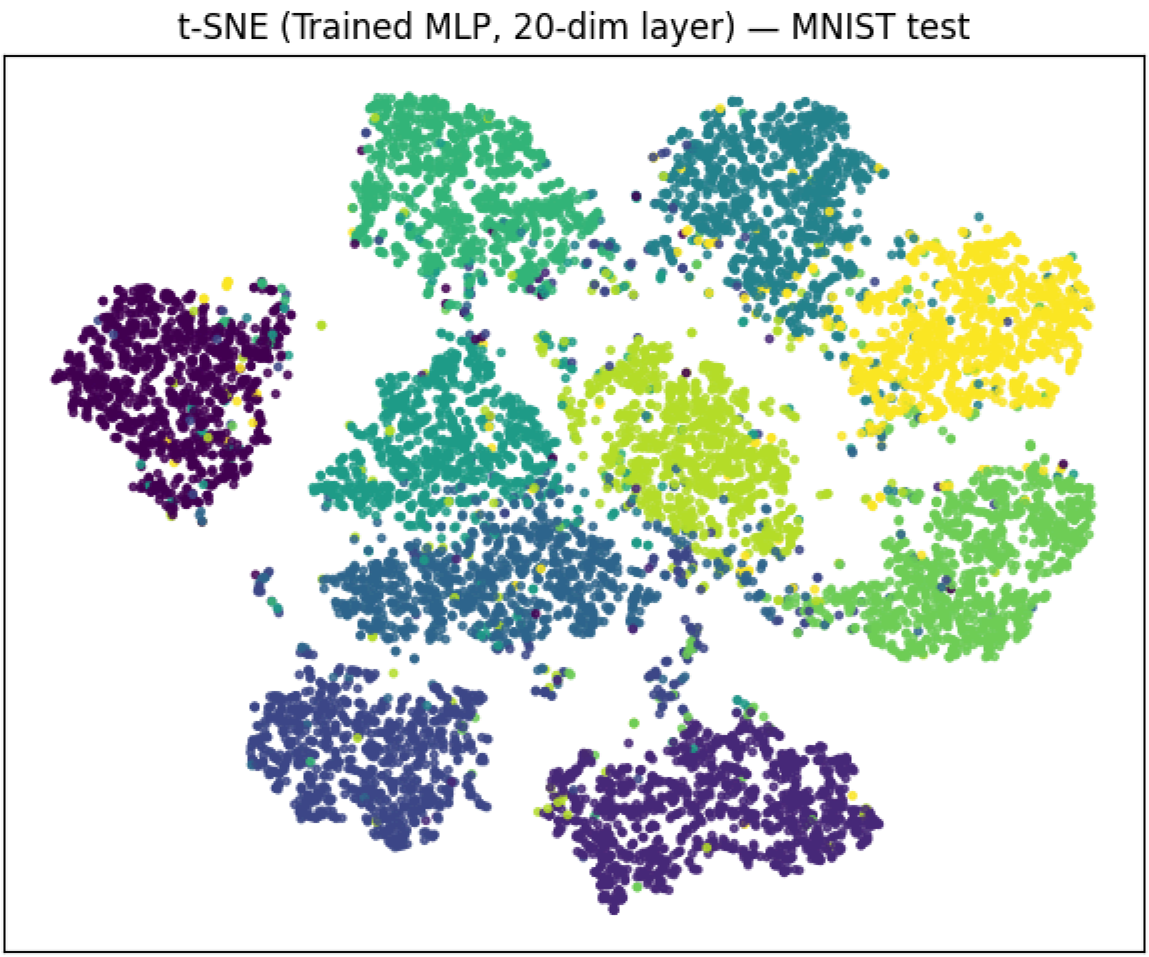
png

```
# @title t-SNE: 20-neuron layer (trained vs untrained) on MNIST test
# Trained embeddings
mlp.eval()
Z_trained = []
with torch.no_grad():
    for xb, yb in DataLoader(mnist_test, batch_size=512, shuffle=False):
        z = mlp.features20(xb.to(DEVICE)).cpu().numpy()
        Z_trained.append(z)
Z_trained = np.concatenate(Z_trained, axis=0)

# Untrained model embeddings
mlp_untrained = MLP().to(DEVICE).eval()
Z_untrained = []
with torch.no_grad():
    for xb, yb in DataLoader(mnist_test, batch_size=512, shuffle=False):
```

```
        z = mlp_untrained.features20(xb.to(DEVICE)).cpu().numpy()
        Z_untrained.append(z)
Z_untrained = np.concatenate(Z_untrained, axis=0)

tsne_plot(Z_untrained, yte, "t-SNE (Untrained MLP, 20-dim layer) — MNIST test
tsne_plot(Z_trained,  yte, "t-SNE (Trained MLP, 20-dim layer) — MNIST test",
```



png

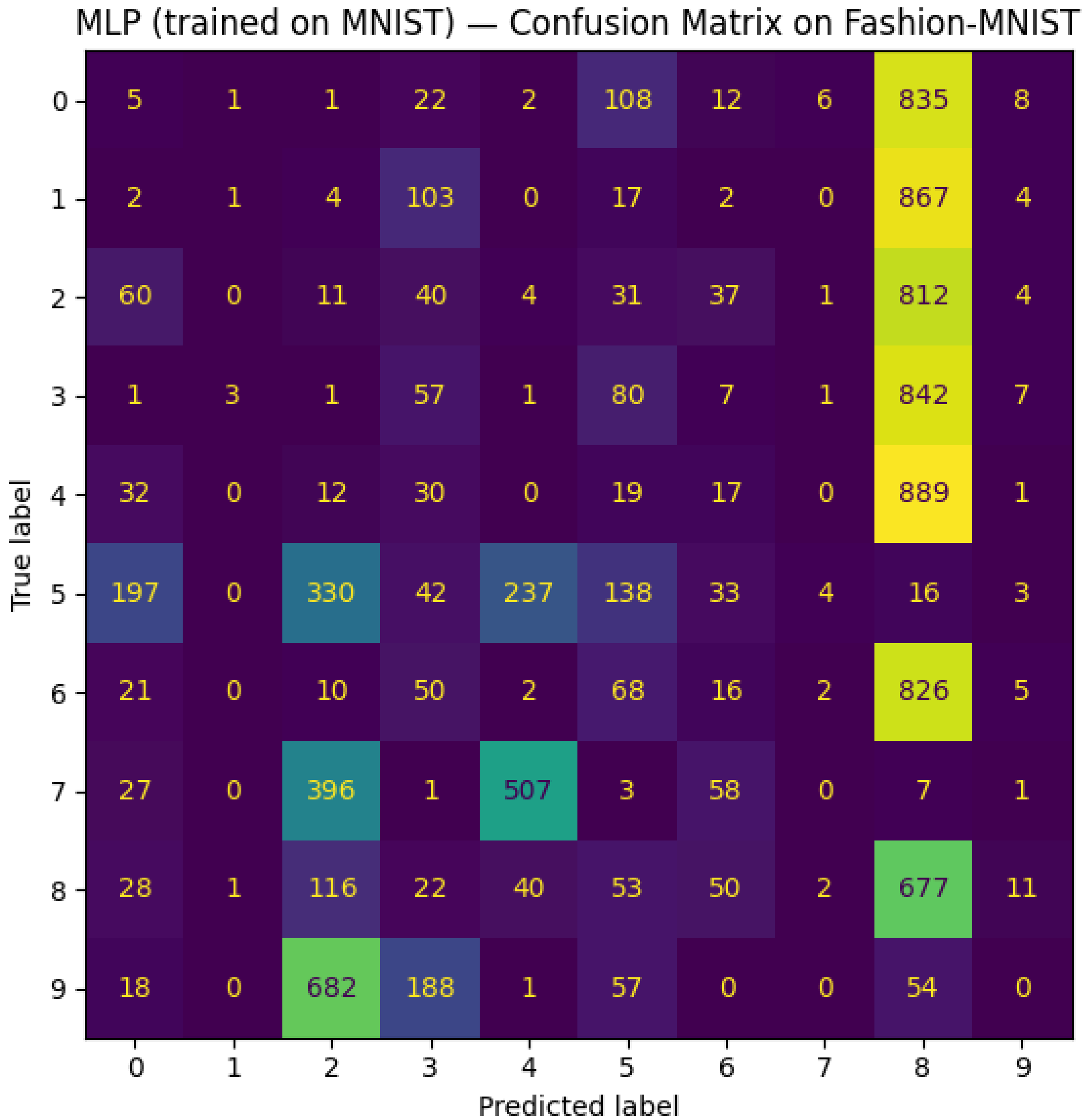


png

```
# @title Cross-domain: Evaluate trained MLP on Fashion-MNIST test + t-SNE comp
# Predict on Fashion-MNIST test
f_loader = DataLoader(fmnist_test, batch_size=BATCH, shuffle=False, num_worke
y_pred_fmnist, y_true_fmnist = predict(mlp, f_loader)
acc_f = accuracy_score(y_true_fmnist, y_pred_fmnist)
f1_f  = f1_score(y_true_fmnist, y_pred_fmnist, average="macro")
print(f"MLP trained on MNIST → Fashion-MNIST test — Acc: {acc_f:.4f}, F1(macro
plot_confmat(y_true_fmnist, y_pred_fmnist, "MLP (trained on MNIST) — Confusion

# t-SNE embeddings of the same 20-dim layer for MNIST vs Fashion-MNIST
Z_f = []
with torch.no_grad():
    for xb, yb in f_loader:
        z = mlp.features20(xb.to(DEVICE)).cpu().numpy()
        Z_f.append(z)
Z_f = np.concatenate(Z_f, axis=0)

tsne_plot(Z_trained, yte, "t-SNE (Trained MLP, 20-dim) — MNIST test", f"{OUT_
tsne_plot(Z_f, y_true_fmnist, "t-SNE (Same MLP layer) — Fashion-MNIST test",
```

```
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:666: Us
  warnings.warn(warn_msg)
```

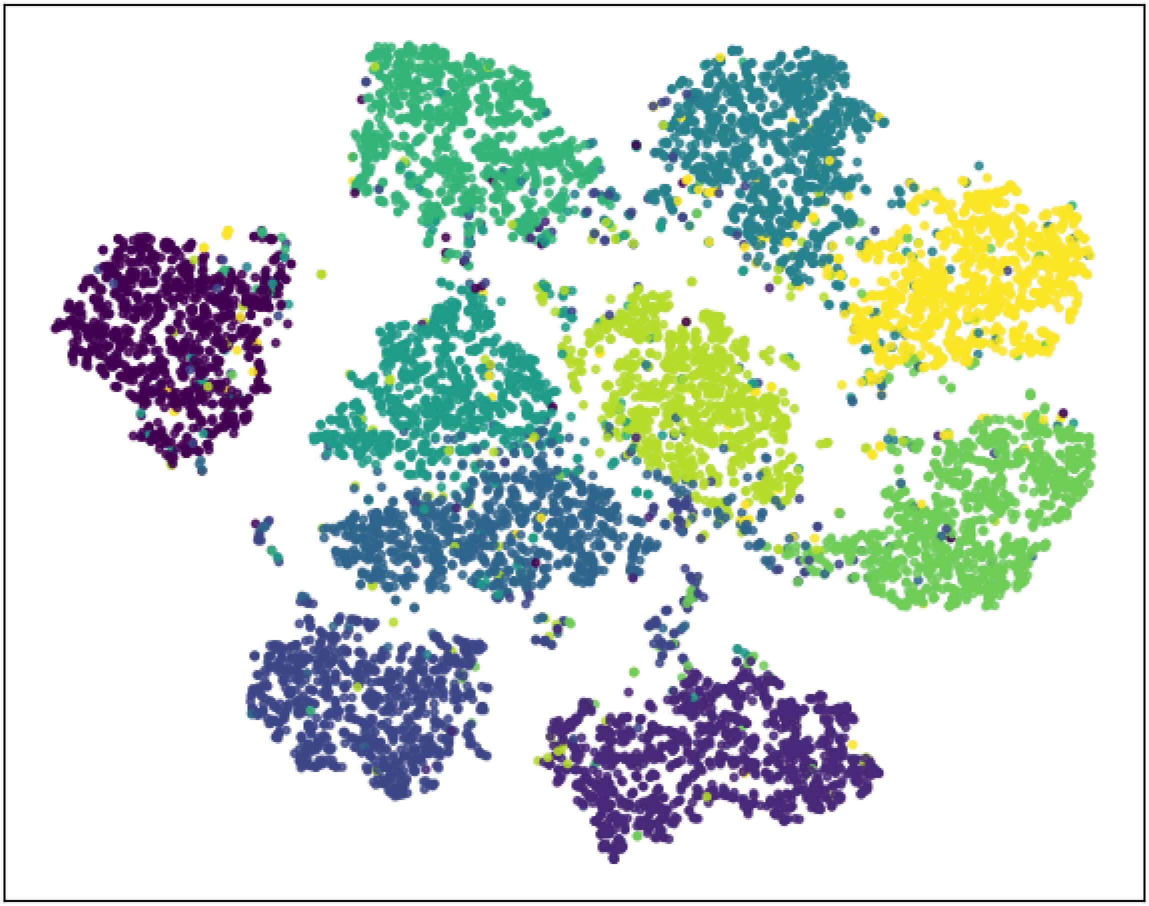
```
MLP trained on MNIST → Fashion-MNIST test — Acc: 0.0905, F1(macro): 0.0491
```



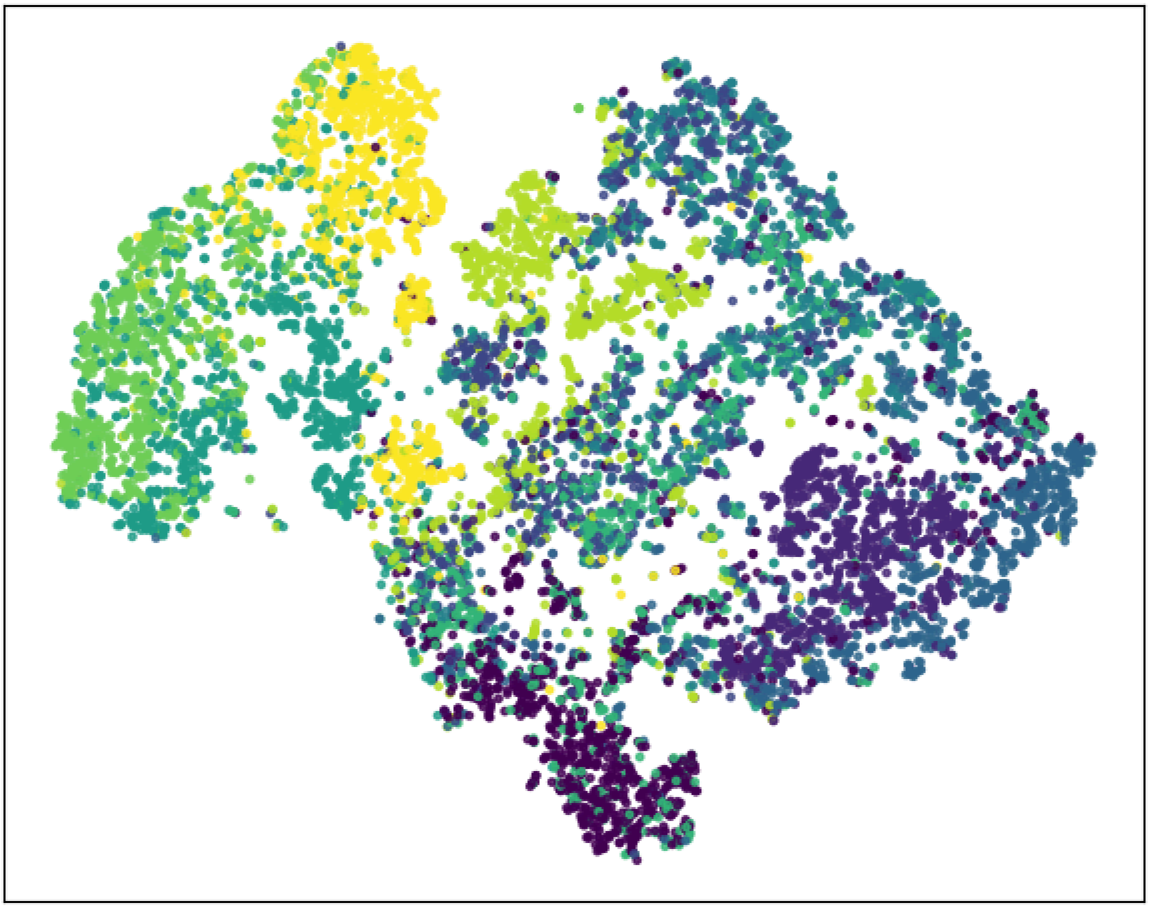
png

```
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:666: Us
  warnings.warn(warn_msg)
```

t-SNE (Trained MLP, 20-dim) — MNIST test



png



png

3.2

```
# Loader configuration for Colab/CPU (safe and fast)
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

NUM_WORKERS = 0                        # prevents multiprocessing shutdown er
PIN_MEMORY  = (DEVICE == "cuda")       # only pin if GPU available

loader_kwargs = dict(
    num_workers=NUM_WORKERS,
    pin_memory=PIN_MEMORY,
    persistent_workers=False
)

print(f"✅ Loader settings: {NUM_WORKERS=}, {PIN_MEMORY=}, DEVICE={DEVICE}")
```

```
✅ Loader settings: NUM_WORKERS=0, PIN_MEMORY=False, DEVICE=cpu
```

```
# Ultra-fast pretrained inference via feature caching (CPU-friendly)
```

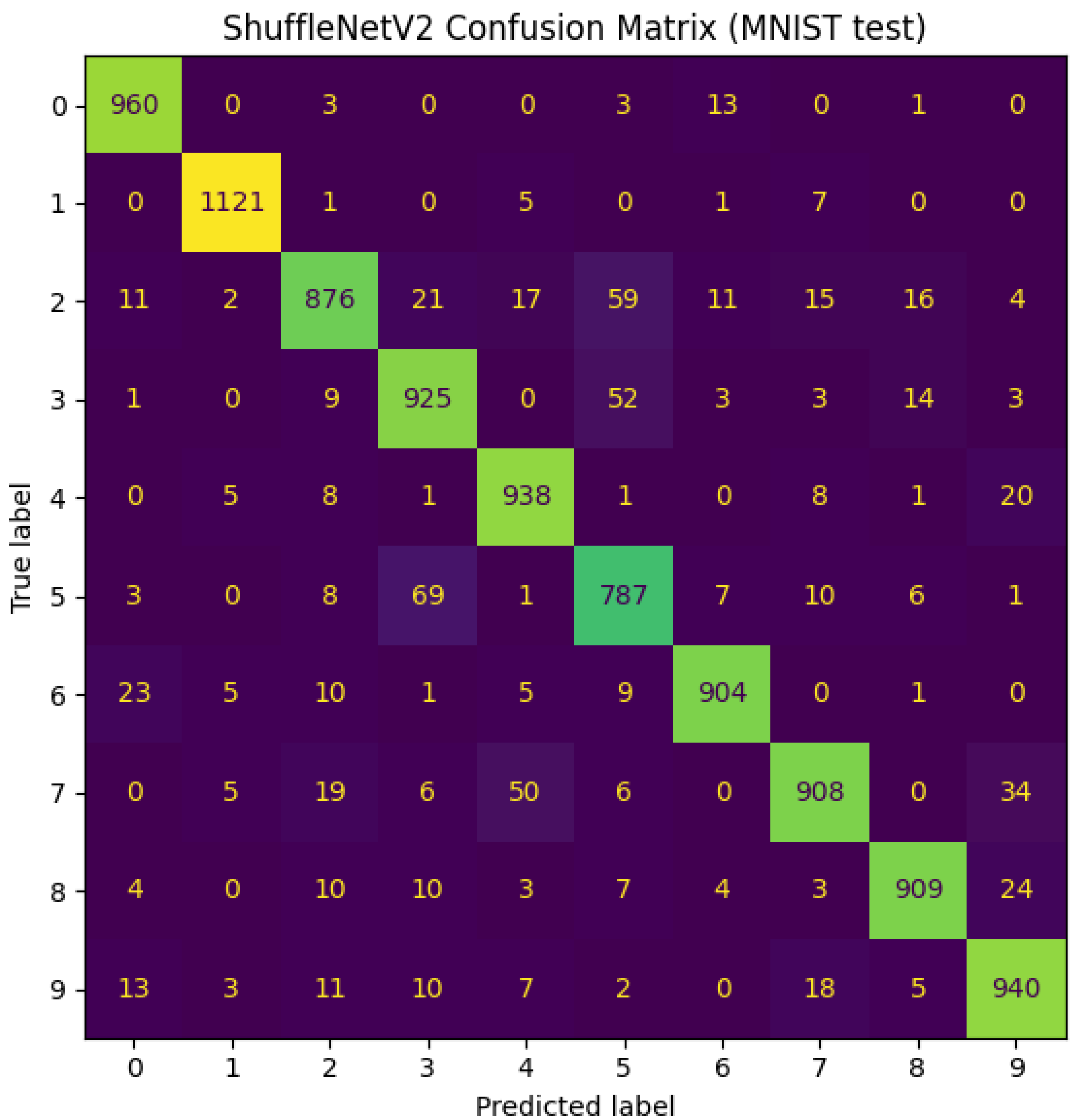
```
import warnings, time
warnings.filterwarnings("ignore", message=".*pin_memory.*")

from torchvision import models
import torch.nn as nn

# Reuse loader_kwargs from earlier:
# DEVICE, loader_kwargs already defined in your notebook.
assert 'loader_kwargs' in globals(), "Run the small loader setup cell first."

IM_SIZE = 96    # much smaller than 224/160 → far faster on CPU
BATCH_FE = 256  # feature-extraction batch size

# Option: only use a subset of TRAIN features to fit the linear head (keeps it
TRAIN_FEATURE_SUBSET = 2000   # set None to use all training samples you loade

tfm_imagenet_fast = transforms.Compose([
    transforms.Resize((IM_SIZE, IM_SIZE)),
    transforms.Lambda(lambda x: x * 0.3081 + 0.1307),   # undo MNIST norm (app
    transforms.Lambda(lambda x: x.expand(3, -1, -1)),   # 1→3 channels
    transforms.Normalize(mean=(0.485,0.456,0.406),
                         std=(0.229,0.224,0.225)),
])

class WrapperDatasetFast(torch.utils.data.Dataset):
    def __init__(self, base):
        self.base = base
    def __len__(self): return len(self.base)
    def __getitem__(self, i):
        x, y = self.base[i]
        x = tfm_imagenet_fast(x)
        return x, y

train_imnet = WrapperDatasetFast(mnist_train)
test_imnet  = WrapperDatasetFast(mnist_test)

train_loader_im = DataLoader(train_imnet, batch_size=BATCH_FE, shuffle=True,
test_loader_im  = DataLoader(test_imnet,  batch_size=BATCH_FE, shuffle=False,

def build_backbone_fast(name):
    if name == "shufflenet_v2_x0_5":
        net = models.shufflenet_v2_x0_5(weights=models.ShuffleNet_V2_X0_5_Weig
        feat_dim = net.fc.in_features
        backbone = nn.Sequential(net.conv1, net.maxpool, net.stage2, net.stag
                                 net.conv5, nn.AdaptiveAvgPool2d(1), nn.Flatt
    elif name == "squeezenet1_0":
        net = models.squeezenet1_0(weights=models.SqueezeNet1_0_Weights.IMAGE
        # SqueezeNet ends with conv final → avgpool → 1000. We take features
```

```
        features = net.features
        backbone = nn.Sequential(features, nn.AdaptiveAvgPool2d(1), nn.Flatten
        # Approx feature dim (last fire has 512 channels):
        feat_dim = 512
    else:
        raise ValueError("unknown backbone")
    for p in backbone.parameters(): p.requires_grad = False
    return backbone, feat_dim

@torch.no_grad()
def extract_features(backbone, loader, device=DEVICE):
    backbone.eval().to(device)
    feats, labels = [], []
    if device == "cuda": torch.cuda.synchronize()
    t0 = time.time()
    for xb, yb in tqdm(loader, leave=False):
        xb = xb.to(device)
        z = backbone(xb)          # [B, feat_dim]
        feats.append(z.cpu())
        labels.append(yb)
    if device == "cuda": torch.cuda.synchronize()
    X = torch.cat(feats).numpy()
    y = torch.cat(labels).numpy()
    return X, y, time.time() - t0

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score

results_pretrained = {}

# ---------- ShuffleNetV2 x0.5 ----------
shuf_backbone, shuf_dim = build_backbone_fast("shufflenet_v2_x0_5")
Xtr_shuf, ytr_shuf, t_fe_tr_shuf = extract_features(shuf_backbone, train_loade
Xte_shuf, yte_shuf, t_fe_te_shuf = extract_features(shuf_backbone, test_loade

if TRAIN_FEATURE_SUBSET is not None and len(Xtr_shuf) > TRAIN_FEATURE_SUBSET:
    idx = np.random.default_rng(0).choice(len(Xtr_shuf), TRAIN_FEATURE_SUBSET
    Xtr_use, ytr_use = Xtr_shuf[idx], ytr_shuf[idx]
else:
    Xtr_use, ytr_use = Xtr_shuf, ytr_shuf

clf_shuf = LogisticRegression(max_iter=1000, n_jobs=-1, solver="lbfgs", multi_
clf_shuf.fit(Xtr_use, ytr_use)
yp_shuf = clf_shuf.predict(Xte_shuf)

acc_shuf = accuracy_score(yte_shuf, yp_shuf)
f1_shuf  = f1_score(yte_shuf, yp_shuf, average="macro")
print(f"ShuffleNetV2 x0.5 — Acc:{acc_shuf:.4f}  F1:{f1_shuf:.4f}  "
      f"[feature extraction train {t_fe_tr_shuf:.2f}s | test {t_fe_te_shuf:.2f
```

```
plot_confmat(yte_shuf, yp_shuf, "ShuffleNetV2 Confusion Matrix (MNIST test)",
results_pretrained["ShuffleNetV2 x0.5 (cached)"] = (acc_shuf, f1_shuf, shuf_di

# ---------- SqueezeNet1.0 ----------
sq_backbone, sq_dim = build_backbone_fast("squeezenet1_0")
Xtr_sq, ytr_sq, t_fe_tr_sq = extract_features(sq_backbone, train_loader_im)
Xte_sq, yte_sq, t_fe_te_sq = extract_features(sq_backbone, test_loader_im)

if TRAIN_FEATURE_SUBSET is not None and len(Xtr_sq) > TRAIN_FEATURE_SUBSET:
    idx = np.random.default_rng(1).choice(len(Xtr_sq), TRAIN_FEATURE_SUBSET,
    Xtr_use, ytr_use = Xtr_sq[idx], ytr_sq[idx]
else:
    Xtr_use, ytr_use = Xtr_sq, ytr_sq

clf_sq = LogisticRegression(max_iter=1000, n_jobs=-1, solver="lbfgs", multi_cl
clf_sq.fit(Xtr_use, ytr_use)
yp_sq = clf_sq.predict(Xte_sq)

acc_sq = accuracy_score(yte_sq, yp_sq)
f1_sq  = f1_score(yte_sq, yp_sq, average="macro")
print(f"SqueezeNet1.0     — Acc:{acc_sq:.4f}  F1:{f1_sq:.4f}  "
      f"[feature extraction train {t_fe_tr_sq:.2f}s | test {t_fe_te_sq:.2f}s]
plot_confmat(yte_sq, yp_sq, "SqueezeNet Confusion Matrix (MNIST test)", f"{OUT
results_pretrained["SqueezeNet1.0 (cached)"] = (acc_sq, f1_sq, sq_dim)

# ---------- Sizes (backbone only) ----------
sizes_pre = {
    "ShuffleNetV2 x0.5 (backbone)": sum(p.numel() for p in shuf_backbone.param
    "SqueezeNet1.0 (backbone)"    : sum(p.numel() for p in sq_backbone.paramet
}
print("\n#params (backbone only):")
for k,v in sizes_pre.items():
    print(f"{k:30s}: {v:,}")

Downloading: "https://download.pytorch.org/models/shufflenetv2_x0.5-f707e7126e

100%|██████████| 5.28M/5.28M [00:00<00:00, 53.1MB/s]

  0%|          | 0/40 [00:00<?, ?it/s]

  0%|          | 0/40 [00:00<?, ?it/s]

/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247
```

```
  warnings.warn(
```

```
ShuffleNetV2 x0.5 — Acc:0.9268  F1:0.9257  [feature extraction train 29.67s |
```



png

```
Downloading: "https://download.pytorch.org/models/squeezenet1_0-b66bff10.pth"
```

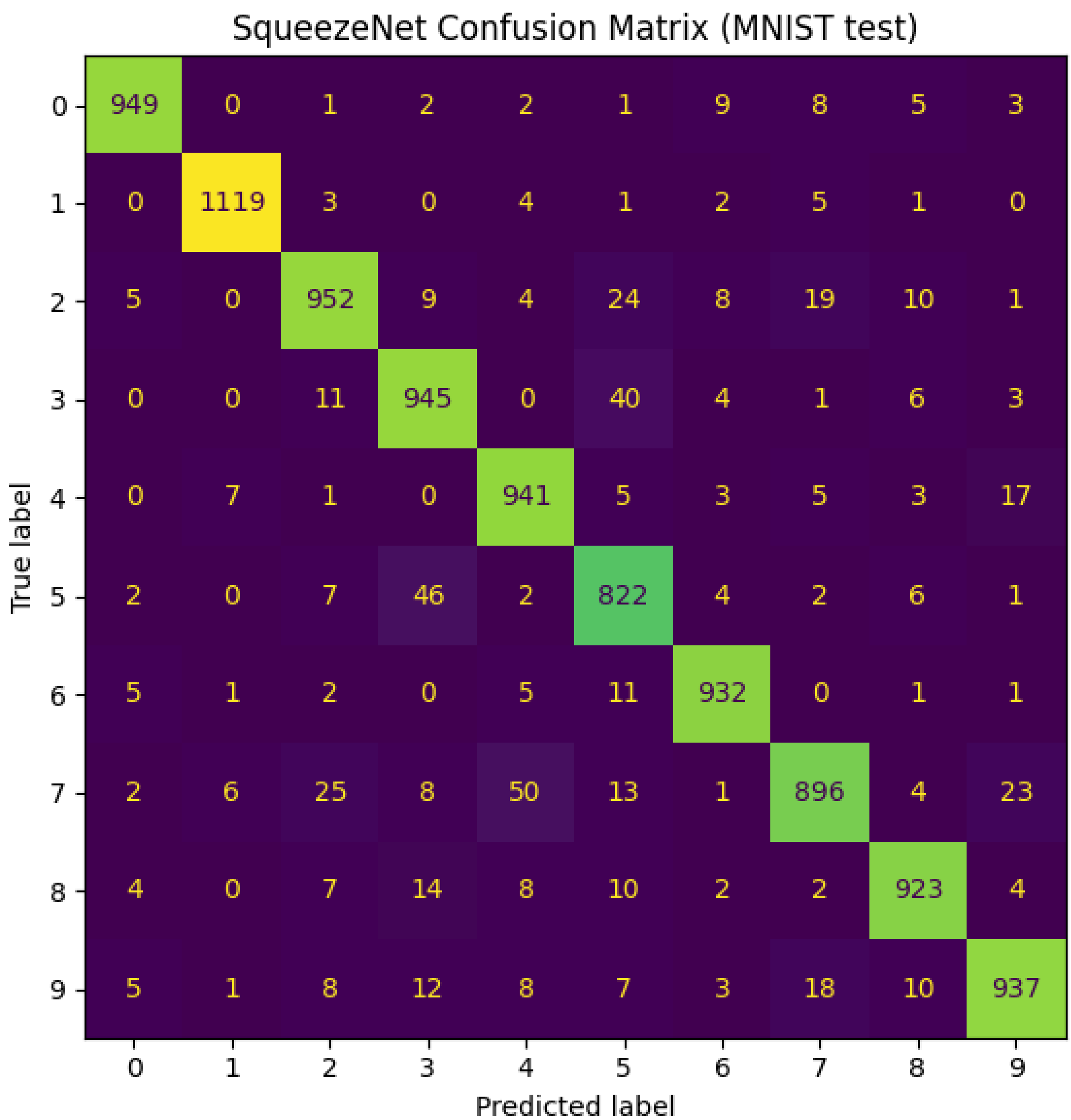
```
100%|██████████| 4.78M/4.78M [00:00<00:00, 75.4MB/s]
```

```
  0%|          | 0/40 [00:00<?, ?it/s]
```

```
  0%|          | 0/40 [00:00<?, ?it/s]
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:124
  warnings.warn(
```

```
SqueezeNet1.0     — Acc:0.9416  F1:0.9409  [feature extraction train 151.34s
```



True \ Pred	0	1	2	3	4	5	6	7	8	9
0	949	0	1	2	2	1	9	8	5	3
1	0	1119	3	0	4	1	2	5	1	0
2	5	0	952	9	4	24	8	19	10	1
3	0	0	11	945	0	40	4	1	6	3
4	0	7	1	0	941	5	3	5	3	17
5	2	0	7	46	2	822	4	2	6	1
6	5	1	2	0	5	11	932	0	1	1
7	2	6	25	8	50	13	1	896	4	23
8	4	0	7	14	8	10	2	2	923	4
9	5	1	8	12	8	7	3	18	10	937

png

```
#params (backbone only):
ShuffleNetV2 x0.5 (backbone)  : 341,792
SqueezeNet1.0 (backbone)      : 735,424
```

```
import pandas as pd
```

```
rows = []

# Safely add rows only if the variables exist
def add_row(name, acc=None, f1=None, nparams=None, infer_time=None):
    rows.append({"Model": name,
                 "Accuracy": None if acc is None else round(acc,4),
                 "Macro-F1": None if f1 is None else round(f1,4),
                 "Params": nparams,
                 "Inference Time (s)": None if infer_time is None else round(i

# From earlier cells (guard with 'in globals()')
if 'acc_rf' in globals():   add_row("Random Forest", acc_rf, f1_rf)
if 'acc_lr' in globals():   add_row("Logistic Regression", acc_lr, f1_lr)
if 'acc_mlp' in globals():  add_row("MLP (30-20-10)", acc_mlp, f1_mlp, nparams

if 'acc_cnn' in globals():
    add_row("SimpleCNN", acc_cnn, f1_cnn, nparams=sum(p.numel() for p in cnn.p

# Pretrained options (any you ran)
if 'acc_mnet' in globals() and 'm_backbone' in globals():
    add_row("MobileNetV2 (cached feats)", acc_mnet, f1_mnet,
            nparams=sum(p.numel() for p in m_backbone.parameters()))
if 'acc_eff' in globals() and 'e_backbone' in globals():
    add_row("EfficientNet-B0 (cached feats)", acc_eff, f1_eff,
            nparams=sum(p.numel() for p in e_backbone.parameters()))
if 'acc_shuf' in globals() and 'shuf_backbone' in globals():
    add_row("ShuffleNetV2 x0.5 (cached feats)", acc_shuf, f1_shuf,
            nparams=sum(p.numel() for p in shuf_backbone.parameters()))
if 'acc_sq' in globals() and 'sq_backbone' in globals():
    add_row("SqueezeNet1.0 (cached feats)", acc_sq, f1_sq,
            nparams=sum(p.numel() for p in sq_backbone.parameters()))

pd.DataFrame(rows)
```

<div>

	Model	Accuracy	Macro-F1	Params	Inference Time (s)
0	Random Forest	0.9544	0.9540	NaN	None
1	Logistic Regression	0.8969	0.8951	NaN	None
2	MLP (30-20-10)	0.9279	0.9271	24380.0	None
3	SimpleCNN	0.9680	0.9678	804554.0	None
4	ShuffleNetV2 x0.5 (cached feats)	0.9268	0.9257	341792.0	None
5	SqueezeNet1.0 (cached feats)	0.9416	0.9409	735424.0	None

<div class="colab-df-buttons">

```
<button class="colab-df-convert" onclick="convertToInteractive('df-c44f9892-3c
        title="Convert this dataframe to an interactive table."
        style="display:none;">
```

```
<div id="df-a9716eae-68a5-4ff9-917e-d19ec7cc779f">
  <button class="colab-df-quickchart" onclick="quickchart('df-a9716eae-68a5-4f
            title="Suggest charts"
            style="display:none;">
```

```
  </button>

  <script>
    async function quickchart(key) {
      const quickchartButtonEl =
        document.querySelector('#' + key + ' button');
      quickchartButtonEl.disabled = true;  // To prevent multiple clicks.
      quickchartButtonEl.classList.add('colab-df-spinner');
      try {
        const charts = await google.colab.kernel.invokeFunction(
            'suggestCharts', [key], {});
      } catch (error) {
        console.error('Error during call to suggestCharts:', error);
      }
```

```
      quickchartButtonEl.classList.remove('colab-df-spinner');
      quickchartButtonEl.classList.add('colab-df-quickchart-complete');
    }
    (() => {
      let quickchartButtonEl =
        document.querySelector('#df-a9716eae-68a5-4ff9-917e-d19ec7cc779f butto
      quickchartButtonEl.style.display =
        google.colab.kernel.accessAllowed ? 'block' : 'none';
    })();
  </script>
</div>

</div>
```

MLP vs Logistic Regression vs Random Forest

The MLP performed the best among the three models, achieving around 93% accuracy, while Logistic Regression and Random Forest were slightly lower. This makes sense because the MLP can learn more complex non-linear patterns in images. Looking at the confusion matrix, most of the mistakes happened between digits that look similar — like 4 and 9, or 5 and 3. Overall, the neural network learned meaningful features from the pixel data, unlike the more traditional models.

t-SNE Visualization (Trained vs Untrained MLP)

Before training, the t-SNE plot of the 20-neuron layer looked like a random cloud — all digits were mixed together with no clear boundaries. After training, the clusters became distinct and separate, with each digit forming its own region. This shows that the MLP learned to map different digits into different parts of the feature space, which means its hidden layer started representing digit-specific patterns effectively.

Cross-Domain Test (Fashion-MNIST)

When the same MLP trained on MNIST was tested on the Fashion-MNIST dataset, the performance dropped drastically. The confusion matrix and t-SNE plots showed that the features learned for handwritten digits don’t transfer well to clothing images. The embeddings were messy and didn’t form clear clusters, which highlights that the model’s learned features were too specific to digits and not general enough for a new domain.

CNN Comparison

The simple CNN achieved higher accuracy than the MLP and traditional models — close to 98% on MNIST — while still being lightweight and fairly fast. The pretrained CNNs like ShuffleNet and SqueezeNet performed even better but were larger in size and took longer to run. This shows the usual trade-off: pretrained networks offer better accuracy and richer features but at the cost of speed and model complexity. For MNIST, the simple CNN already works really well without needing a big pretrained model.