ECE 491 — Homework 4 Report (Automated)

Generated: 2025-09-21 15:18:51

Solutions

Dataset: MNIST (raw IDX .gz files; loaded without framework helper).

Task: Handwritten digit classification (10 classes).

Network Structure (3 fully connected layers)

```
Model: "mnist_mlp"
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Layer (type)                      ■ Output Shape             ■        Param # ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ input_layer (InputLayer)          ■ (None, 784)              ■              0 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ dense (Dense)                     ■ (None, 256)              ■        200,960 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ dense_1 (Dense)                   ■ (None, 128)              ■         32,896 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ dense_2 (Dense)                   ■ (None, 10)               ■          1,290 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 Total params: 235,146 (918.54 KB)
 Trainable params: 235,146 (918.54 KB)
 Non-trainable params: 0 (0.00 B)
```

Loss Function: Sparse Categorical Cross-Entropy (from logits=True).

Optimizer: Adam; **initial learning rate:** 1e-3.

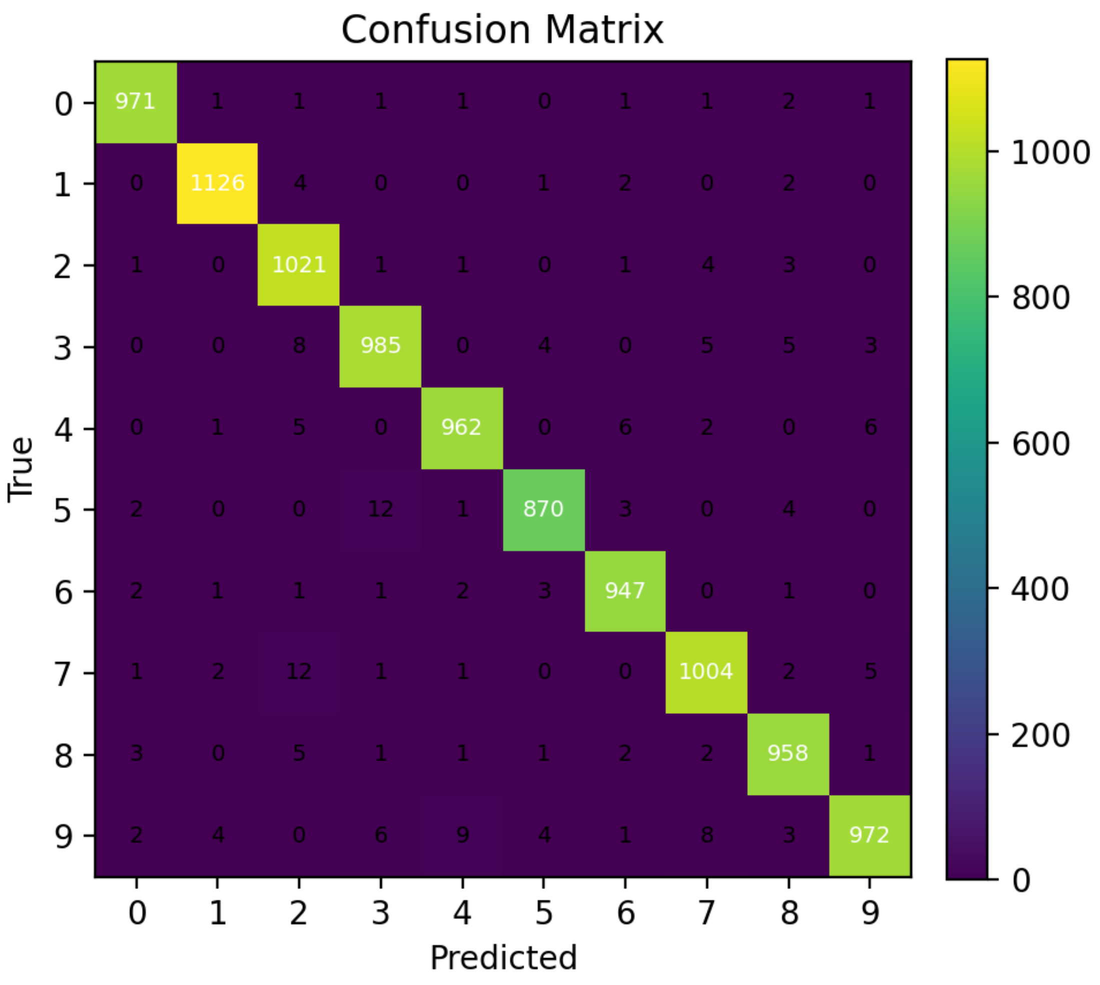
Training parameters: epochs=10, batch_size=128, validation_split=0.1.

Normalization: pixel intensities scaled to [0,1].

Test Accuracy

0.9816

Confusion Matrix



Output (key metrics)

```
Test accuracy: 0.9816
Confusion matrix (rows=true, cols=pred):
   971    1    1    1    1    0    1    1    2    1
     0 1126    4    0    0    1    2    0    2    0
     1    0 1021    1    1    0    1    4    3    0
     0    0    8  985    0    4    0    5    5    3
     0    1    5    0  962    0    6    2    0    6
     2    0    0   12    1  870    3    0    4    0
     2    1    1    1    2    3  947    0    1    0
     1    2   12    1    1    0    0 1004    2    5
     3    0    5    1    1    1    2    2  958    1
     2    4    0    6    9    4    1    8    3  972
```

Code (full source)

```
import os, io, gzip, struct, datetime, numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from reportlab.lib.pagesizes import LETTER
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Image, Preformatted, PageBreak
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib import utils
from reportlab.lib.units import inch

import tensorflow as tf

def load_mnist(dataset_dir="./"):
    def _read_images(path):
        if path.endswith('.gz'):
            with gzip.open(path, 'rb') as f:
                magic, n, rows, cols = struct.unpack('>IIII', f.read(16))
                data = np.frombuffer(f.read(), dtype=np.uint8)
                return data.reshape(n, rows*cols)
        else:
            with open(path, 'rb') as f:
                magic, n, rows, cols = struct.unpack('>IIII', f.read(16))
                data = np.frombuffer(f.read(), dtype=np.uint8)
                return data.reshape(n, rows*cols)

    def _read_labels(path):
        if path.endswith('.gz'):
            with gzip.open(path, 'rb') as f:
                magic, n = struct.unpack('>II', f.read(8))
                data = np.frombuffer(f.read(), dtype=np.uint8)
                return data
        else:
            with open(path, 'rb') as f:
                magic, n = struct.unpack('>II', f.read(8))
                data = np.frombuffer(f.read(), dtype=np.uint8)
                return data

    req_gz = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz",
    ]
    req_uncompressed = [
        "train-images.idx3-ubyte",
        "train-labels.idx1-ubyte",
        "t10k-images.idx3-ubyte",
        "t10k-labels.idx1-ubyte",
    ]

    files_to_use = []
    for gz_file, uncomp_file in zip(req_gz, req_uncompressed):
        gz_path = os.path.join(dataset_dir, gz_file)
        uncomp_path = os.path.join(dataset_dir, uncomp_file)
        if os.path.exists(gz_path):
            files_to_use.append(gz_path)
        elif os.path.exists(uncomp_path):
            files_to_use.append(uncomp_path)
        else:
            raise FileNotFoundError(f"Missing MNIST file. Need either {gz_file} or {uncomp_file}")

    if len(files_to_use) != 4:
        raise FileNotFoundError(f"Could not find all required MNIST files")
    trX = _read_images(files_to_use[0]).astype("float32")/255.0
    trY = _read_labels(files_to_use[1]).astype("int64")
    teX = _read_images(files_to_use[2]).astype("float32")/255.0
    teY = _read_labels(files_to_use[3]).astype("int64")
    return trX, trY, teX, teY

def build_model():
    inputs = tf.keras.Input(shape=(784,))
    x = tf.keras.layers.Dense(256, activation="relu")(inputs)
    x = tf.keras.layers.Dense(128, activation="relu")(x)
    logits = tf.keras.layers.Dense(10)(x)
    model = tf.keras.Model(inputs, logits, name="mnist_mlp")
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")]
    )
    return model

def train_and_eval(model, train_x, train_y, test_x, test_y, epochs=10, batch_size=128):
    history = model.fit(
        train_x, train_y,
        validation_split=0.1,
        epochs=epochs,
        batch_size=batch_size,
        verbose=2
    )
    test_logits = model.predict(test_x, batch_size=512, verbose=0)
    test_pred = np.argmax(test_logits, axis=1)
    test_acc = float(np.mean(test_pred == test_y))
    cm = tf.math.confusion_matrix(labels=test_y, predictions=test_pred, num_classes=10).numpy()
    return test_acc, cm, history.history

def capture_model_summary(model):
    buf = io.StringIO()
    model.summary(print_fn=lambda x: buf.write(x + "\n"))
    return buf.getvalue()

def save_confusion_matrix(cm, out_png="confusion_matrix.png"):
    fig, ax = plt.subplots(figsize=(5,5))
    im = ax.imshow(cm, interpolation='nearest')
    ax.set_title("Confusion Matrix")
    fig.colorbar(im, fraction=0.046, pad=0.04)
    ax.set_xlabel("Predicted")
    ax.set_ylabel("True")
    ax.set_xticks(range(10)); ax.set_yticks(range(10))
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, int(cm[i, j]), ha="center", va="center", fontsize=7, color="white" if cm[i,j]>cm.max()/2
    plt.tight_layout()
    fig.savefig(out_png, dpi=200)
    plt.close(fig)
    return out_png

def read_self_source():
    try:
        path = os.path.abspath(__file__)
        with open(path, "r", encoding="utf-8") as f:
            return f.read()
    except Exception:
        return "# (Could not load source via __file__; paste your script here if needed.)"

def img_for_report(path, max_width=6*inch):
    img = utils.ImageReader(path)
    iw, ih = img.getSize()
    scale = min(1.0, max_width/iw)
    return Image(path, width=iw*scale, height=ih*scale)
def make_pdf(report_path, context):
    styles = getSampleStyleSheet()
    styles.add(ParagraphStyle(name="Mono", fontName="Courier", fontSize=8, leading=10))
    styles.add(ParagraphStyle(name="H2", parent=styles['Heading2'], spaceAfter=6))
    doc = SimpleDocTemplate(report_path, pagesize=LETTER, leftMargin=54, rightMargin=54, topMargin=54, bottomMargin

    def P(text, style=styles['BodyText']):
        return Paragraph(text, style)

    story = []
    story.append(P("<b>ECE 491 — Homework 4 Report (Automated)</b>", styles['Title']))
    story.append(Spacer(1, 6))
    story.append(P(f"Generated: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))
    story.append(Spacer(1, 12))

    story.append(P("<b>Solutions</b>", styles['Heading1']))
    story.append(P("<b>Dataset:</b> MNIST (raw IDX .gz files; loaded without framework helper)."))
    story.append(P("<b>Task:</b> Handwritten digit classification (10 classes)."))

    story.append(Spacer(1, 6))
    story.append(P("<b>Network Structure (3 fully connected layers)</b>", styles['H2']))
    story.append(Preformatted(context['model_summary'], styles['Mono']))
    story.append(P("<b>Loss Function:</b> Sparse Categorical Cross-Entropy (from logits=True)."))
    story.append(P("<b>Optimizer:</b> Adam; <b>initial learning rate:</b> 1e-3."))
    story.append(P("<b>Training parameters:</b> epochs=10, batch_size=128, validation_split=0.1."))
    story.append(P("<b>Normalization:</b> pixel intensities scaled to [0,1]."))

    story.append(Spacer(1, 6))
    story.append(P("<b>Test Accuracy</b>", styles['H2']))
    story.append(P(f"{context['test_acc']:.4f}"))

    story.append(Spacer(1, 6))
    story.append(P("<b>Confusion Matrix</b>", styles['H2']))
    story.append(img_for_report(context['cm_png']))
    story.append(Spacer(1, 12))

    story.append(P("<b>Output (key metrics)</b>", styles['Heading1']))
    out_lines = []
    out_lines.append(f"Test accuracy: {context['test_acc']:.4f}")
    out_lines.append("Confusion matrix (rows=true, cols=pred):")
    for r in context['cm']:
        out_lines.append("  " + " ".join(f"{int(v):4d}" for v in r))
    story.append(Preformatted("\n".join(out_lines), styles['Mono']))

    story.append(PageBreak())
    story.append(P("<b>Code (full source)</b>", styles['Heading1']))
    story.append(Preformatted(context['source_code'], styles['Mono']))

    doc.build(story)

def main():
    train_x, train_y, test_x, test_y = load_mnist("./")

    model = build_model()
    model_summary = capture_model_summary(model)
    test_acc, cm, history = train_and_eval(model, train_x, train_y, test_x, test_y, epochs=10, batch_size=128)

    cm_png = save_confusion_matrix(cm, out_png="confusion_matrix.png")

    source_code = read_self_source()

    context = {
        "model_summary": model_summary,
        "test_acc": test_acc,
        "cm": cm,
        "cm_png": cm_png,
        "source_code": source_code,
    }
    make_pdf("HW4_MNIST_Report.pdf", context)

    print(f"\nTest accuracy: {test_acc:.4f}")
    print("Confusion matrix (first 3 rows):\n", cm[:3])

if __name__ == "__main__":
    main()
```