AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
International Credit Hours Programs (ICHEP)
Mechatronics and Automation

Machine Vision (CSE480)

Lab 4 Report

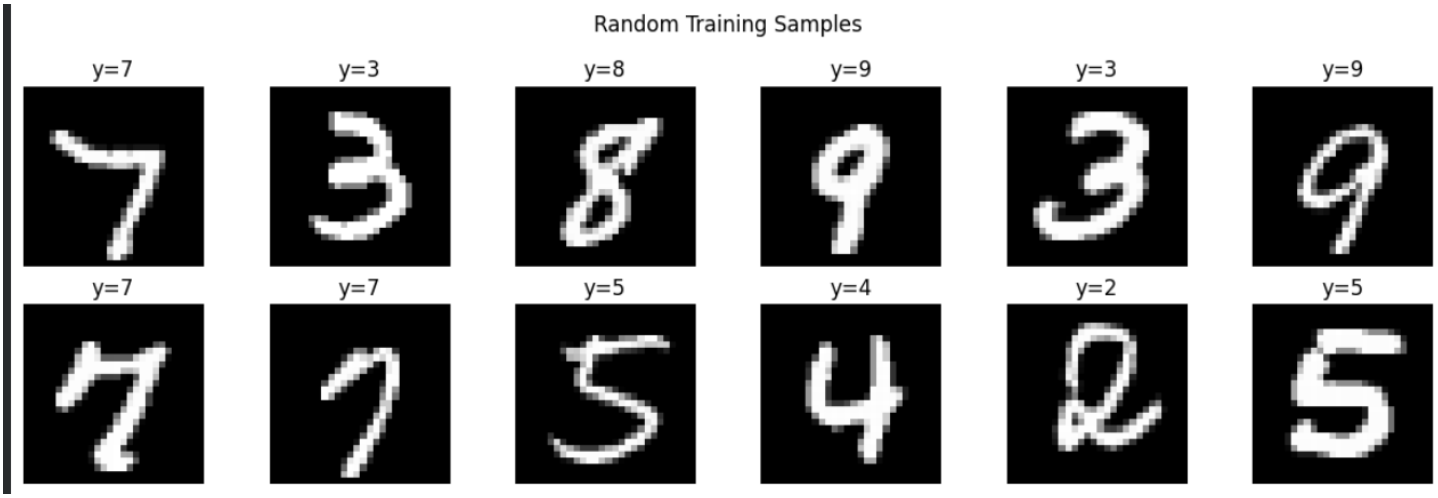
Name	ID	Section
Mahmoud Elsayd Abdelqader Labib Eldwakhly	21P0017	1

Submitted to Dr Hossam Hassan & Eng. Dina Zakaria

Fall 2025

Task 1 :

Dataset Exploration :

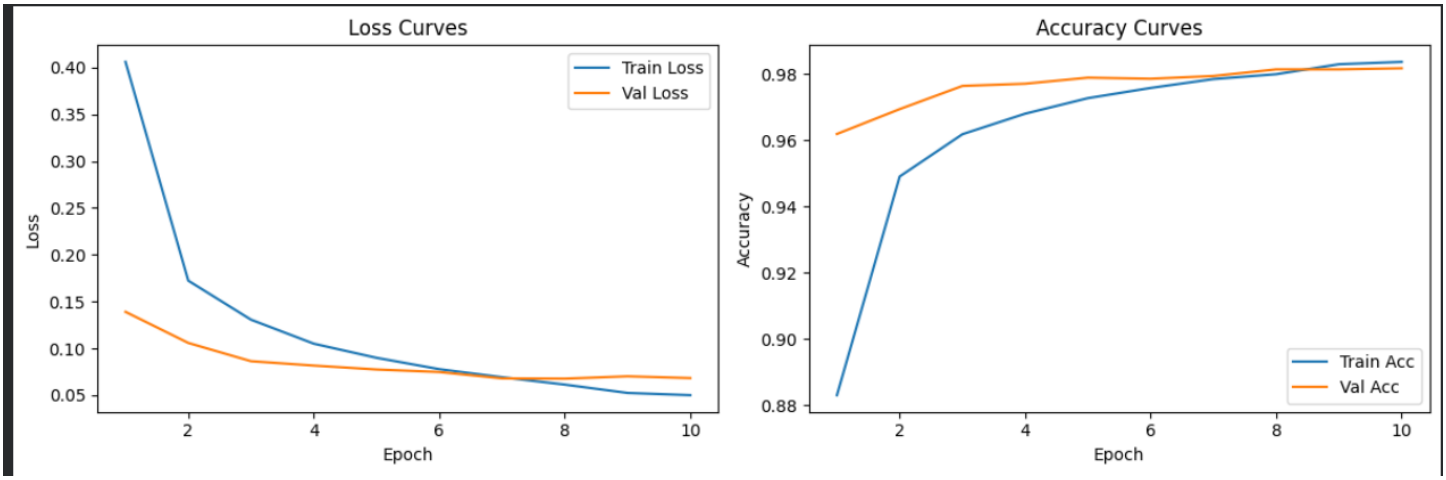


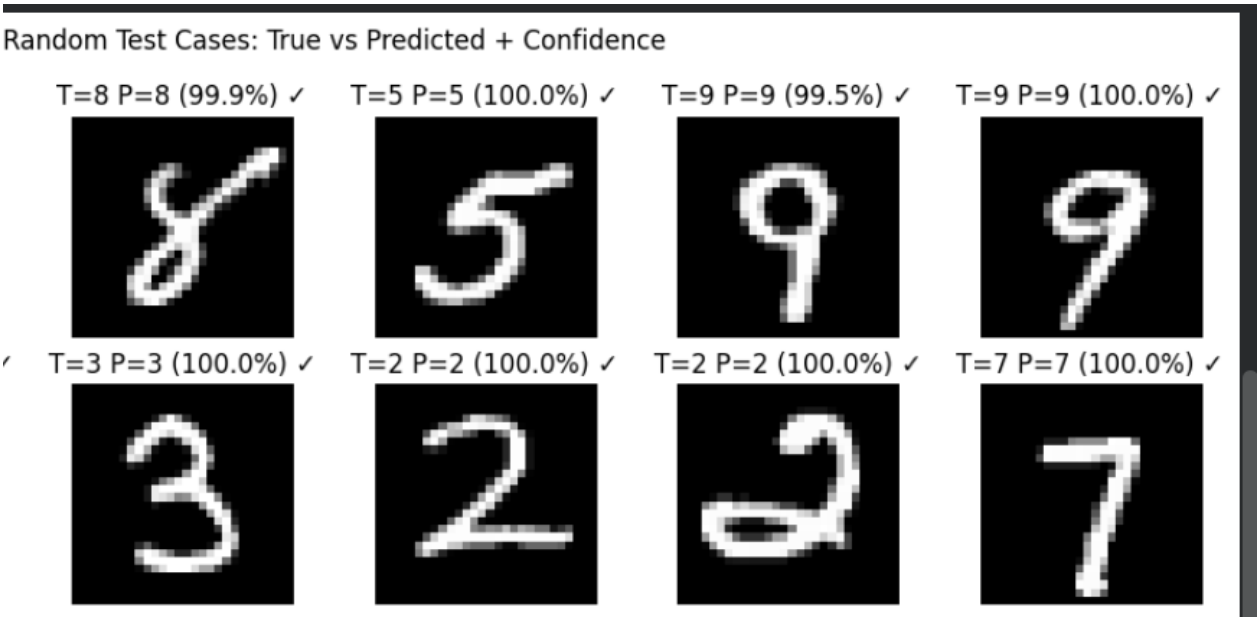

```
=== MODEL SUMMARY ===
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 128)	100,480
dropout_1 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8,256
dense_5 (Dense)	(None, 10)	650

```
Total params: 109,386 (427.29 KB)
Trainable params: 109,386 (427.29 KB)
Non-trainable params: 0 (0.00 B)
```

```python
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(784,)),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
```






```
Model:
- Dense-only network: 128(ReLU) + Dropout + 64(ReLU) + 10(Softmax).

Training (10 epochs):
- Final Train Loss: 0.0500 | Final Val Loss: 0.0683
- Final Train Acc : 0.9836 | Final Val Acc : 0.9817

Testing:
- Test Loss: 0.0697 | Test Accuracy: 0.9780
```

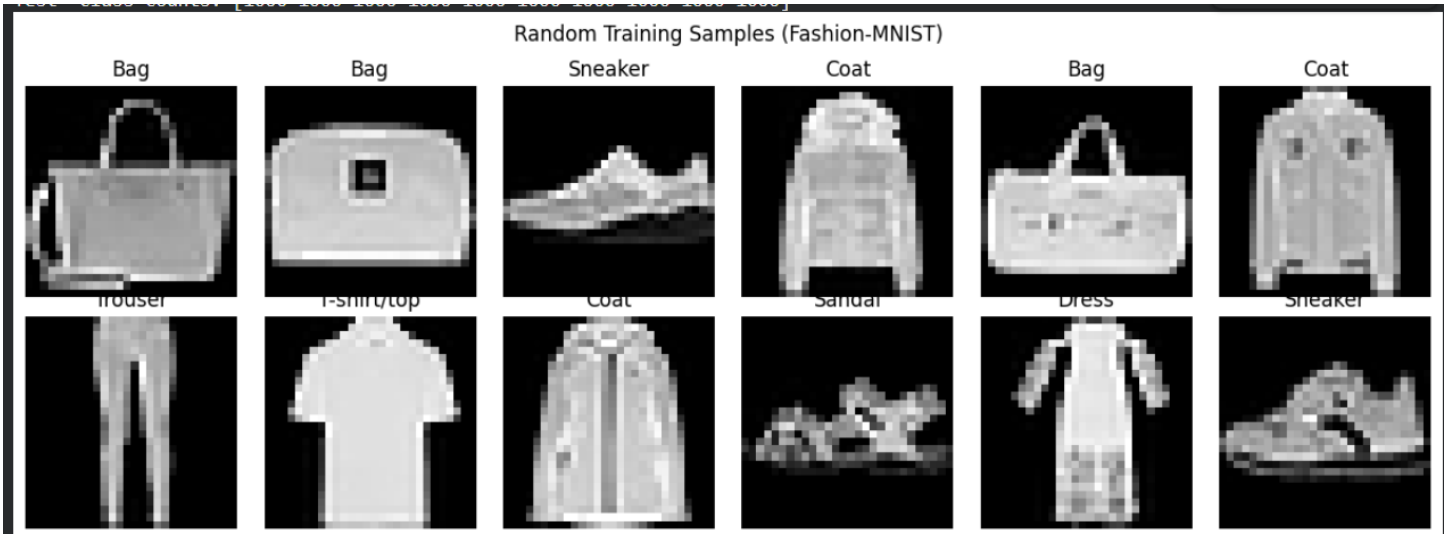
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

tf.random.set_seed(42)
np.random.seed(42)

# ----------------------------
# 1) Load MNIST
# ----------------------------
(x_train_img, y_train), (x_test_img, y_test) = tf.keras.datasets.mnist.load_data()

print("=== DATASET OVERVIEW ===")
print("Train images shape:", x_train_img.shape, " Train labels shape:", y_train.shape)
print("Test  images shape:", x_test_img.shape,  " Test  labels shape:", y_test.shape)
print("Pixel range (train):", (x_train_img.min(), x_train_img.max()))
print("Pixel range (test): ", (x_test_img.min(), x_test_img.max()))
print("Classes:", np.unique(y_train))

# ----------------------------
# 2) Explore dataset
# ----------------------------
def plot_class_distribution(labels, title):
    counts = np.bincount(labels, minlength=10)
    plt.figure(figsize=(8,3))
    plt.bar(range(10), counts)
    plt.xticks(range(10))
    plt.xlabel("Digit class")
    plt.ylabel("Count")
    plt.title(title)
    plt.show()
    return counts

train_counts = plot_class_distribution(y_train, "Train Class Distribution")
test_counts  = plot_class_distribution(y_test,  "Test Class Distribution")

print("Train class counts:", train_counts)
print("Test  class counts:", test_counts)

def show_samples(images, labels, n=12, title="Random Samples"):
    idx = np.random.choice(len(images), n, replace=False)
    cols = 6
    rows = int(np.ceil(n / cols))
    plt.figure(figsize=(12, 2*rows))
    for i, k in enumerate(idx):
        plt.subplot(rows, cols, i+1)
        plt.imshow(images[k], cmap="gray")
        plt.title(f"y={labels[k]}")
        plt.axis("off")
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

show_samples(x_train_img, y_train, n=12, title="Random Training Samples")

# ----------------------------
# 3) Preprocess: normalize + flatten
# ----------------------------
x_train = (x_train_img.astype("float32") / 255.0).reshape(-1, 28*28)
x_test  = (x_test_img.astype("float32")  / 255.0).reshape(-1, 28*28)

# ----------------------------
# 4) Build Dense-only model
# ----------------------------
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(784,)),
```

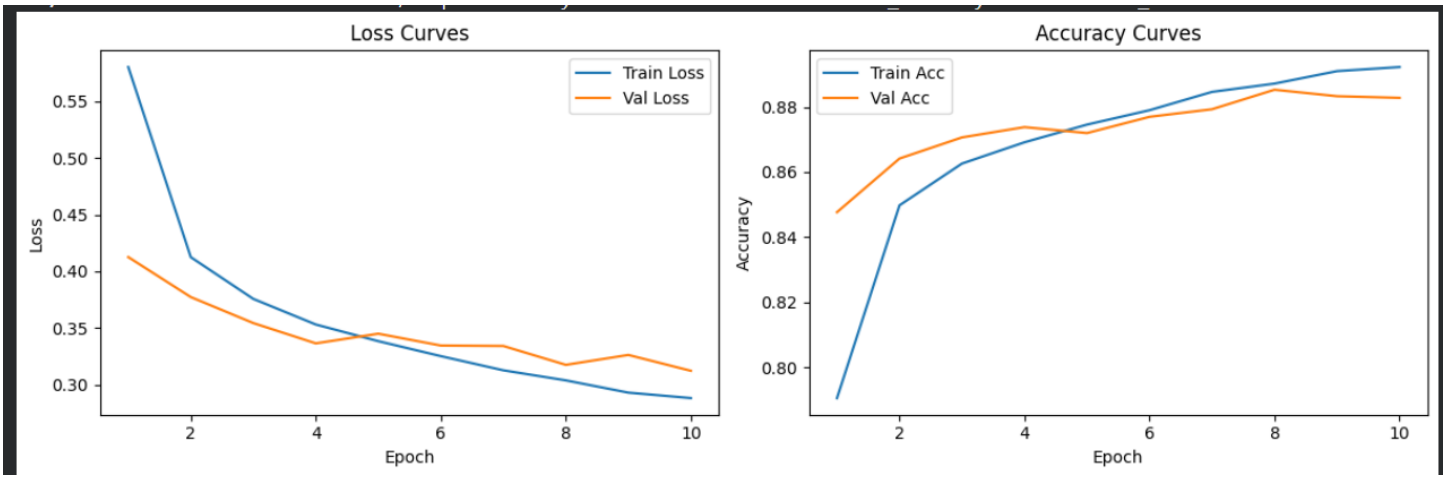
```
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-3),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

print("\n=== MODEL SUMMARY ===")
model.summary()

# ----------------------------
# 5) Train for 10 epochs
# ----------------------------
EPOCHS = 10
history = model.fit(
    x_train, y_train,
    validation_split=0.1,
    epochs=EPOCHS,
    batch_size=128,
    verbose=1
)

# ----------------------------
# 6) Plot training curves
# ----------------------------
def plot_training_curves(hist):
    h = hist.history
    ep = range(1, len(h["loss"]) + 1)
    plt.figure(figsize=(12,4))

    plt.subplot(1,2,1)
    plt.plot(ep, h["loss"], label="Train Loss")
    plt.plot(ep, h["val_loss"], label="Val Loss")
    plt.xlabel("Epoch"); plt.ylabel("Loss")
    plt.title("Loss Curves"); plt.legend()

    plt.subplot(1,2,2)
    plt.plot(ep, h["accuracy"], label="Train Acc")
    plt.plot(ep, h["val_accuracy"], label="Val Acc")
    plt.xlabel("Epoch"); plt.ylabel("Accuracy")
    plt.title("Accuracy Curves"); plt.legend()

    plt.tight_layout()
    plt.show()

plot_training_curves(history)

# ----------------------------
# 7) Evaluate on test set
# ----------------------------
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print("\n=== TEST PERFORMANCE ===")
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")

# ----------------------------
# 8) Test cases (predictions) with output
# ----------------------------
def predict_test_cases(model, x_test_flat, x_test_images, y_test, n_cases=12):
    idx = np.random.choice(len(x_test_flat), n_cases, replace=False)
    probs = model.predict(x_test_flat[idx], verbose=0)
```

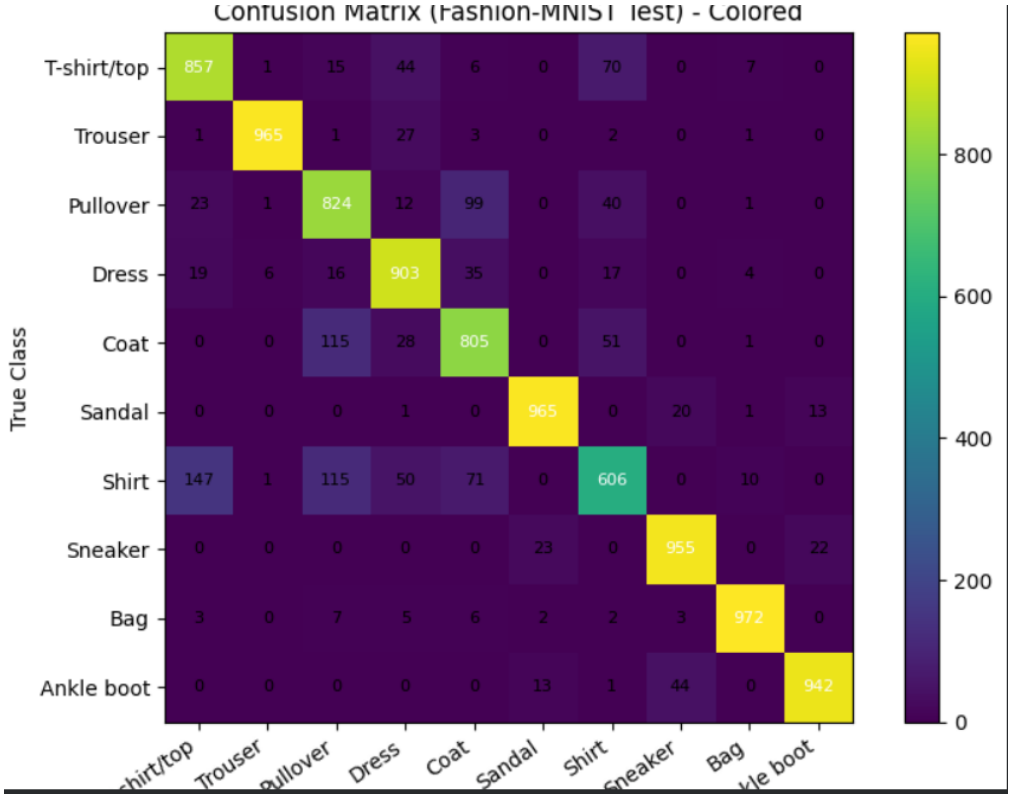
```
    preds = np.argmax(probs, axis=1)
    confs = np.max(probs, axis=1)

    cols = 6
    rows = int(np.ceil(n_cases / cols))
    plt.figure(figsize=(12, 2*rows))
    for i, k in enumerate(idx):
        plt.subplot(rows, cols, i+1)
        plt.imshow(x_test_images[k], cmap="gray")
        correct = (preds[i] == y_test[k])
        mark = "✓" if correct else "✗"
        plt.title(f"T={y_test[k]} P={preds[i]} ({confs[i]*100:.1f}%) {mark}")
        plt.axis("off")
    plt.suptitle("Random Test Cases: True vs Predicted + Confidence")
    plt.tight_layout()
    plt.show()

    # Also print a small table-like output
    print("\nSample Predictions (index, true, pred, confidence):")
    for i, k in enumerate(idx):
        print(f"- idx={k:5d} | true={y_test[k]} | pred={preds[i]} | conf={confs[i]:.4f}")

predict_test_cases(model, x_test, x_test_img, y_test, n_cases=12)

# ----------------------------
# 9) Confusion Matrix (no sklearn needed)
# ----------------------------
def confusion_matrix_np(y_true, y_pred, num_classes=10):
    cm = np.zeros((num_classes, num_classes), dtype=np.int32)
    for t, p in zip(y_true, y_pred):
        cm[t, p] += 1
    return cm

all_probs = model.predict(x_test, verbose=0)
y_pred = np.argmax(all_probs, axis=1)

cm = confusion_matrix_np(y_test, y_pred, num_classes=10)

plt.figure(figsize=(6,5))
plt.imshow(cm, cmap="Blues")
plt.title("Confusion Matrix (Test)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.xticks(range(10))
plt.yticks(range(10))
plt.colorbar()
plt.tight_layout()
plt.show()

# Print a few most confused pairs
off_diag = cm.copy()
np.fill_diagonal(off_diag, 0)
pairs = []
for i in range(10):
    for j in range(10):
        if i != j and off_diag[i, j] > 0:
            pairs.append((off_diag[i, j], i, j))
pairs.sort(reverse=True)

print("\n=== MOST COMMON CONFUSIONS (true -> predicted) ===")
for c, t, p in pairs[:10]:
    print(f"- {t} -> {p}: {c} times")

# ----------------------------
# 10) Show misclassified examples
# ----------------------------
```

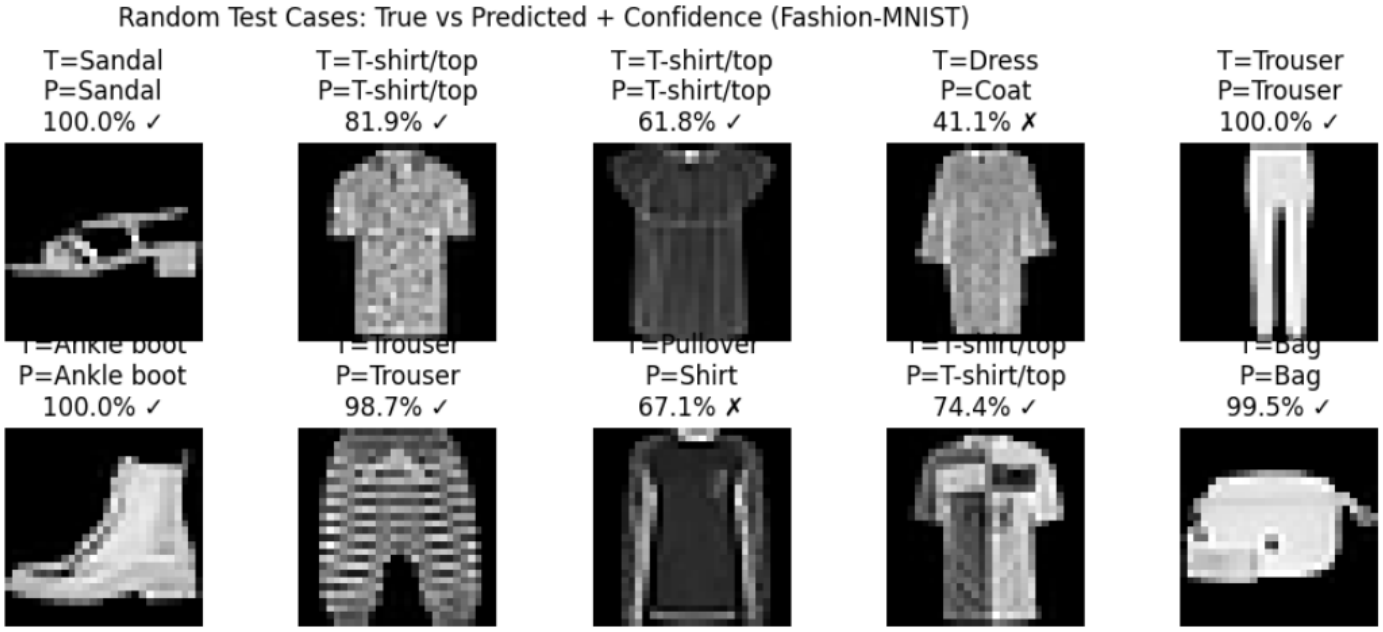
```python
def show_misclassified(x_test_images, y_true, y_pred, probs, n=12):
    wrong_idx = np.where(y_true != y_pred)[0]
    if len(wrong_idx) == 0:
        print("No misclassifications found (rare).")
        return
    pick = np.random.choice(wrong_idx, size=min(n, len(wrong_idx)), replace=False)

    cols = 6
    rows = int(np.ceil(len(pick) / cols))
    plt.figure(figsize=(12, 2*rows))
    for i, k in enumerate(pick):
        plt.subplot(rows, cols, i+1)
        plt.imshow(x_test_images[k], cmap="gray")
        conf = np.max(probs[k])
        plt.title(f"T={y_true[k]} P={y_pred[k]} ({conf*100:.1f}%)")
        plt.axis("off")
    plt.suptitle("Misclassified Test Examples")
    plt.tight_layout()
    plt.show()

show_misclassified(x_test_img, y_test, y_pred, all_probs, n=12)

# ----------------------------
# 11) Small report
# ----------------------------
train_loss_last = history.history["loss"][-1]
val_loss_last   = history.history["val_loss"][-1]
train_acc_last  = history.history["accuracy"][-1]
val_acc_last    = history.history["val_accuracy"][-1]

print("\n" + "="*60)
print("SMALL DETAILED REPORT")
print("="*60)
print("Exploration:")
print("- MNIST has 60,000 train images and 10,000 test images, each 28x28 grayscale.")
print("- We displayed random samples and plotted class distributions (counts per digit).")
print("\nModel:")
print("- Dense-only network: 128(ReLU) + Dropout + 64(ReLU) + 10(Softmax).")
print("\nTraining (10 epochs):")
print(f"- Final Train Loss: {train_loss_last:.4f} | Final Val Loss: {val_loss_last:.4f}")
print(f"- Final Train Acc : {train_acc_last:.4f} | Final Val Acc : {val_acc_last:.4f}")
print("\nTesting:")
print(f"- Test Loss: {test_loss:.4f} | Test Accuracy: {test_acc:.4f}")
print("\nTest Cases & Errors:")
print("- We predicted random test cases and showed True vs Predicted with confidence.")
print("- We plotted a confusion matrix + printed most common confusions.")
print("- We displayed some misclassified examples to inspect error patterns.")
print("="*60)
```

Task 2 :


Random Training Samples (Fashion-MNIST)
Bag
Bag
Sneaker
Coat
Bag
Coat
Trouser
T-shirt/top
Coat
Sandal
Dress
Sneaker


Loss Curves
Train Loss
Val Loss
Loss
Epoch
Accuracy Curves
Train Acc
Val Acc
Accuracy
Epoch


Confusion Matrix (Fashion-MNIST Test) - Colored
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
True Class
857 1 15 44 6 0 70 0 7 0
1 965 1 27 3 0 2 0 1 0
23 1 824 12 99 0 40 0 1 0
19 6 16 903 35 0 17 0 4 0
0 0 115 28 805 0 51 0 1 0
0 0 0 1 0 965 0 20 1 13
147 1 115 50 71 0 606 0 10 0
0 0 0 0 0 23 0 955 0 22
3 0 7 5 6 2 2 3 972 0
0 0 0 0 0 13 1 44 0 942
800
600
400
200
0




```
Preprocessing:
- Normalized pixel values to [0, 1].
- Flattened each image from 28x28 to a 784-length vector.

Model (Dense-only):
- Dense(256, ReLU) + Dropout(0.3) + Dense(128, ReLU) + Dense(10, Softmax).

Training (10 epochs):
- Final Train Loss: 0.2882 | Final Val Loss: 0.3122
- Final Train Acc : 0.8923 | Final Val Acc : 0.8828

Testing:
- Test Loss: 0.3378 | Test Accuracy: 0.8794
```

```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

tf.random.set_seed(42)
np.random.seed(42)

# ----------------------------
# 1) Load Fashion-MNIST
# ----------------------------
(x_train_img, y_train), (x_test_img, y_test) = tf.keras.datasets.fashion_mnist.load_data()

class_names = [
    "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
    "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
]

print("=== DATASET OVERVIEW (Fashion-MNIST) ===")
print("Train images shape:", x_train_img.shape, " Train labels shape:", y_train.shape)
print("Test  images shape:", x_test_img.shape,  " Test  labels shape:", y_test.shape)
print("Pixel range (train):", (x_train_img.min(), x_train_img.max()))
print("Pixel range (test): ", (x_test_img.min(), x_test_img.max()))
print("Classes (0-9):", list(range(10)))
print("Class names:", class_names)

# ----------------------------
# 2) Explore dataset (distribution + sample images)
# ----------------------------
def plot_class_distribution(labels, title):
    counts = np.bincount(labels, minlength=10)
    plt.figure(figsize=(10,3))
    plt.bar(range(10), counts)
    plt.xticks(range(10), class_names, rotation=35, ha="right")
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.title(title)
    plt.tight_layout()
    plt.show()
    return counts

train_counts = plot_class_distribution(y_train, "Train Class Distribution (Fashion-MNIST)")
test_counts  = plot_class_distribution(y_test,  "Test Class Distribution (Fashion-MNIST)")

print("Train class counts:", train_counts)
print("Test  class counts:", test_counts)

def show_samples(images, labels, n=12, title="Random Samples"):
    idx = np.random.choice(len(images), n, replace=False)
    cols = 6
    rows = int(np.ceil(n / cols))
    plt.figure(figsize=(12, 2.2*rows))
    for i, k in enumerate(idx):
        plt.subplot(rows, cols, i+1)
        plt.imshow(images[k], cmap="gray")
        plt.title(class_names[labels[k]])
        plt.axis("off")
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

show_samples(x_train_img, y_train, n=12, title="Random Training Samples (Fashion-MNIST)")

# ----------------------------
# 3) Normalize + Flatten
# ----------------------------
x_train = (x_train_img.astype("float32") / 255.0).reshape(-1, 28*28)
x_test  = (x_test_img.astype("float32")  / 255.0).reshape(-1, 28*28)

# ----------------------------
# 4) Build Dense-only model
# ----------------------------
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(784,)),
    tf.keras.layers.Dense(256, activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss="sparse_categorical_crossentropy",
```

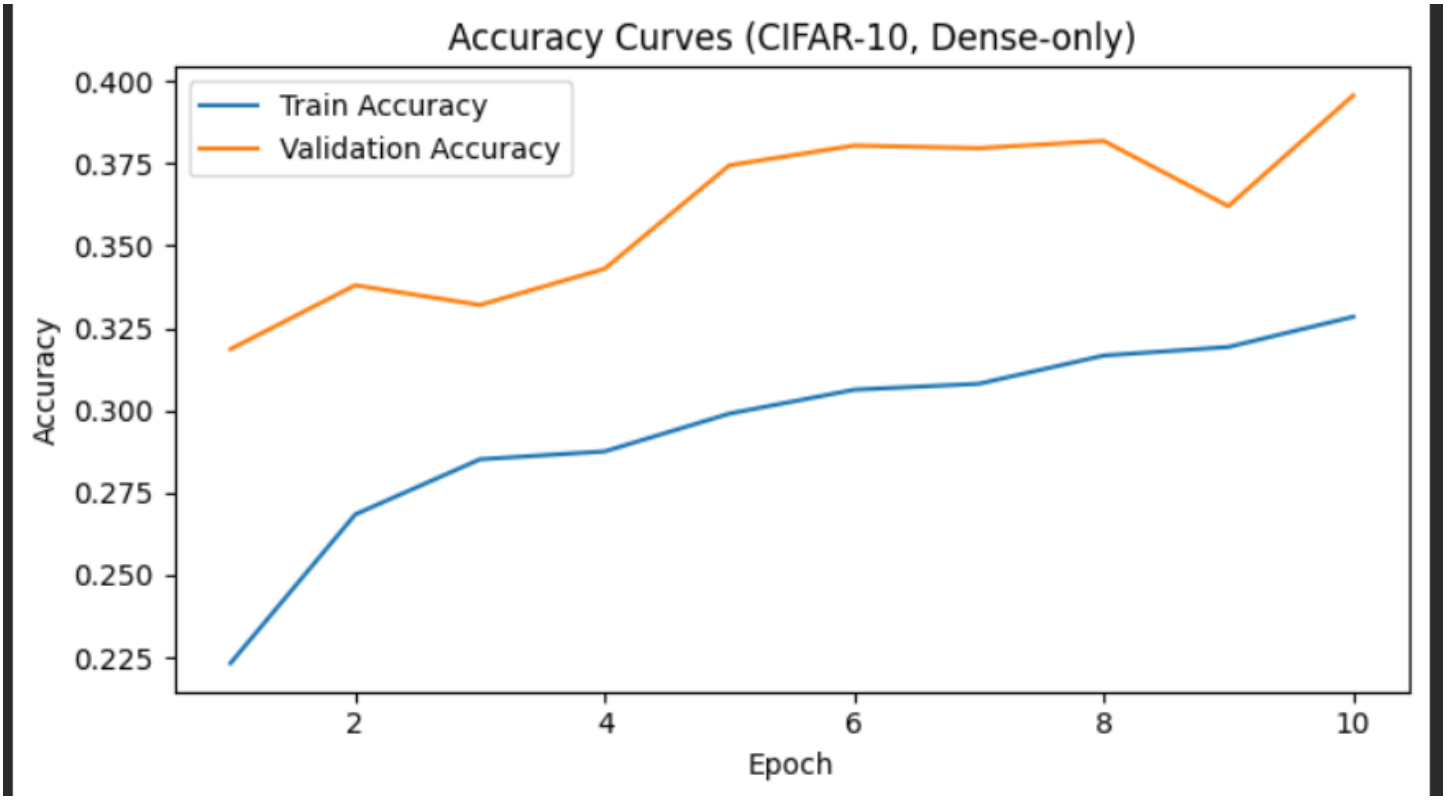
```
    metrics=["accuracy"]
)

print("\n=== MODEL SUMMARY ===")
model.summary()

# ----------------------------
# 5) Train for 10 epochs
# ----------------------------
EPOCHS = 10
history = model.fit(
    x_train, y_train,
    epochs=EPOCHS,
    batch_size=128,
    validation_split=0.1,
    verbose=1
)

# ----------------------------
# 6) Plot training curves (loss + accuracy)
# ----------------------------
def plot_training_curves(hist):
    h = hist.history
    ep = range(1, len(h["loss"]) + 1)
    plt.figure(figsize=(12,4))

    plt.subplot(1,2,1)
    plt.plot(ep, h["loss"], label="Train Loss")
    plt.plot(ep, h["val_loss"], label="Val Loss")
    plt.xlabel("Epoch"); plt.ylabel("Loss")
    plt.title("Loss Curves"); plt.legend()

    plt.subplot(1,2,2)
    plt.plot(ep, h["accuracy"], label="Train Acc")
    plt.plot(ep, h["val_accuracy"], label="Val Acc")
    plt.xlabel("Epoch"); plt.ylabel("Accuracy")
    plt.title("Accuracy Curves"); plt.legend()

    plt.tight_layout()
    plt.show()

plot_training_curves(history)

# ----------------------------
# 7) Test evaluation
# ----------------------------
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print("\n=== TEST PERFORMANCE ===")
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")

# ----------------------------
# 8) Predictions + Confusion Matrix (colored)
# ----------------------------
all_probs = model.predict(x_test, verbose=0)
y_pred = np.argmax(all_probs, axis=1)

def confusion_matrix_np(y_true, y_pred, num_classes=10):
    cm = np.zeros((num_classes, num_classes), dtype=np.int32)
    for t, p in zip(y_true, y_pred):
        cm[t, p] += 1
    return cm

cm = confusion_matrix_np(y_test, y_pred, num_classes=10)

plt.figure(figsize=(8,6))
plt.imshow(cm, cmap="viridis")  # colored heatmap
plt.title("Confusion Matrix (Fashion-MNIST Test) - Colored")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.xticks(range(10), class_names, rotation=35, ha="right")
plt.yticks(range(10), class_names)
plt.colorbar()

# write numbers on the heatmap for readability
thresh = cm.max() * 0.6
for i in range(10):
    for j in range(10):
        plt.text(j, i, cm[i, j],
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black",
                 fontsize=8)
```

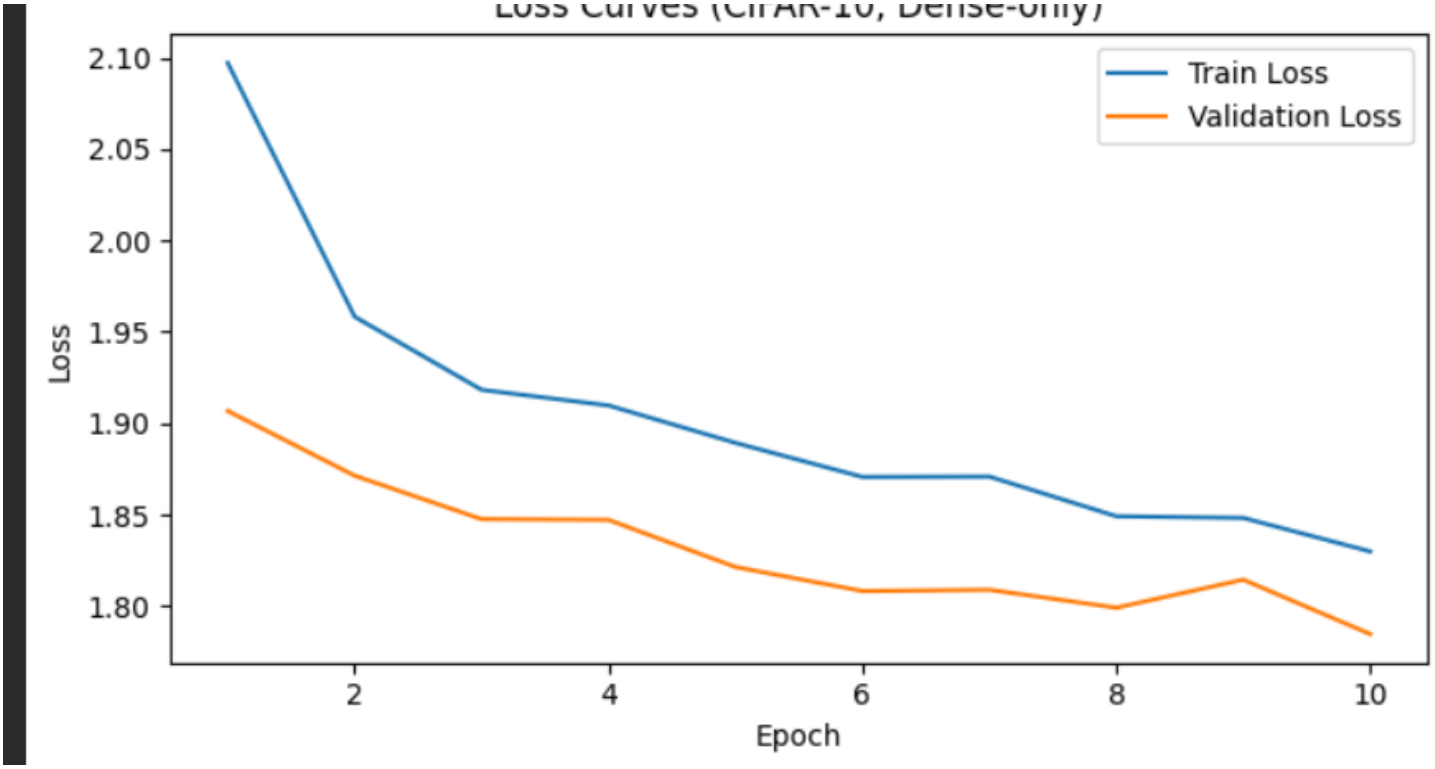
```
plt.tight_layout()
plt.show()

# Print most common confusions (true -> predicted)
off_diag = cm.copy()
np.fill_diagonal(off_diag, 0)
pairs = []
for i in range(10):
    for j in range(10):
        if i != j and off_diag[i, j] > 0:
            pairs.append((off_diag[i, j], i, j))
pairs.sort(reverse=True)

print("\n=== MOST COMMON CONFUSIONS (true -> predicted) ===")
for c, t, p in pairs[:10]:
    print(f"- {class_names[t]} -> {class_names[p]}: {c} times")

# ----------------------------
# 9) Show misclassified examples
# ----------------------------
def show_misclassified(x_test_images, y_true, y_pred, probs, n=12):
    wrong_idx = np.where(y_true != y_pred)[0]
    if len(wrong_idx) == 0:
        print("No misclassifications found (rare).")
        return
    pick = np.random.choice(wrong_idx, size=min(n, len(wrong_idx)), replace=False)

    cols = 6
    rows = int(np.ceil(len(pick) / cols))
    plt.figure(figsize=(12, 2.2*rows))
    for i, k in enumerate(pick):
        plt.subplot(rows, cols, i+1)
        plt.imshow(x_test_images[k], cmap="gray")
        conf = np.max(probs[k])
        plt.title(f"T={class_names[y_true[k]]}\nP={class_names[y_pred[k]]}\n({conf*100:.1f}%)")
        plt.axis("off")
    plt.suptitle("Misclassified Test Examples (Fashion-MNIST)")
    plt.tight_layout()
    plt.show()

show_misclassified(x_test_img, y_test, y_pred, all_probs, n=12)

# ----------------------------
# 10) Random test cases (visual output + confidence)
# ----------------------------
def predict_test_cases(model, x_test_flat, x_test_images, y_test, n_cases=12):
    idx = np.random.choice(len(x_test_flat), n_cases, replace=False)
    probs = model.predict(x_test_flat[idx], verbose=0)
    preds = np.argmax(probs, axis=1)
    confs = np.max(probs, axis=1)

    cols = 6
    rows = int(np.ceil(n_cases / cols))
    plt.figure(figsize=(12, 2.2*rows))
    for i, k in enumerate(idx):
        plt.subplot(rows, cols, i+1)
        plt.imshow(x_test_images[k], cmap="gray")
        correct = (preds[i] == y_test[k])
        mark = "✓" if correct else "✗"
        plt.title(f"T={class_names[y_test[k]]}\nP={class_names[preds[i]]}\n{confs[i]*100:.1f}% {mark}")
        plt.axis("off")
    plt.suptitle("Random Test Cases: True vs Predicted + Confidence (Fashion-MNIST)")
    plt.tight_layout()
    plt.show()

    print("\nSample Predictions (index, true, pred, confidence):")
    for i, k in enumerate(idx):
        print(f"- idx={k:5d} | true={class_names[y_test[k]]:12s} | pred={class_names[preds[i]]:12s} | conf={confs[i]:.4f}")

predict_test_cases(model, x_test, x_test_img, y_test, n_cases=12)

# ----------------------------
# 11) Small detailed report
# ----------------------------
train_loss_last = history.history["loss"][-1]
val_loss_last   = history.history["val_loss"][-1]
train_acc_last  = history.history["accuracy"][-1]
val_acc_last    = history.history["val_accuracy"][-1]

print("\n" + "="*60)
print("SMALL DETAILED REPORT (Task 2: Fashion-MNIST)")
```

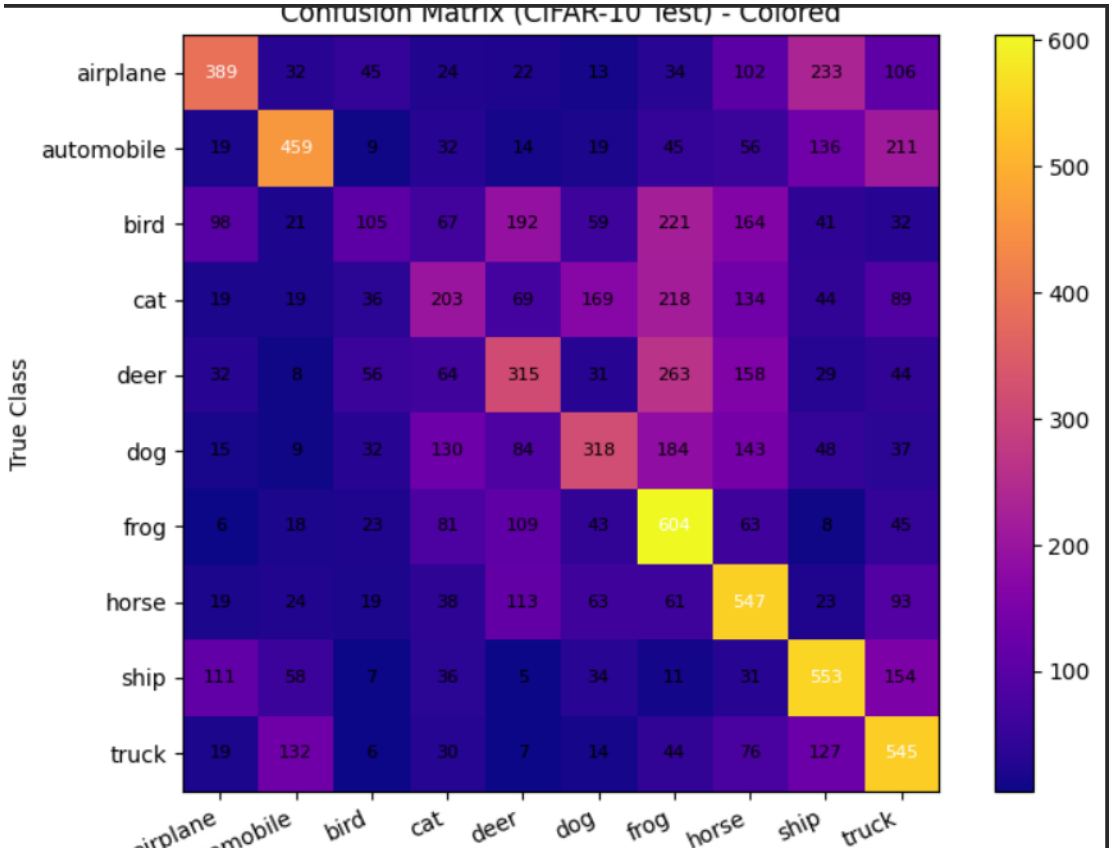
```
print("="*60)
print("Exploration:")
print("- Loaded Fashion-MNIST: 60,000 train and 10,000 test grayscale images (28x28).")
print("- Printed shapes/pixel ranges, plotted class distributions, and showed sample images.")
print("\nPreprocessing:")
print("- Normalized pixel values to [0, 1].")
print("- Flattened each image from 28x28 to a 784-length vector.")
print("\nModel (Dense-only):")
print("- Dense(256, ReLU) + Dropout(0.3) + Dense(128, ReLU) + Dense(10, Softmax).")
print("\nTraining (10 epochs):")
print(f"- Final Train Loss: {train_loss_last:.4f} | Final Val Loss: {val_loss_last:.4f}")
print(f"- Final Train Acc : {train_acc_last:.4f} | Final Val Acc : {val_acc_last:.4f}")
print("\nTesting:")
print(f"- Test Loss: {test_loss:.4f} | Test Accuracy: {test_acc:.4f}")
print("\nError Analysis:")
print("- Built a colored confusion matrix (with counts in each cell).")
print("- Printed the most frequent confusions and visualized misclassified examples.")
print("="*60)
```

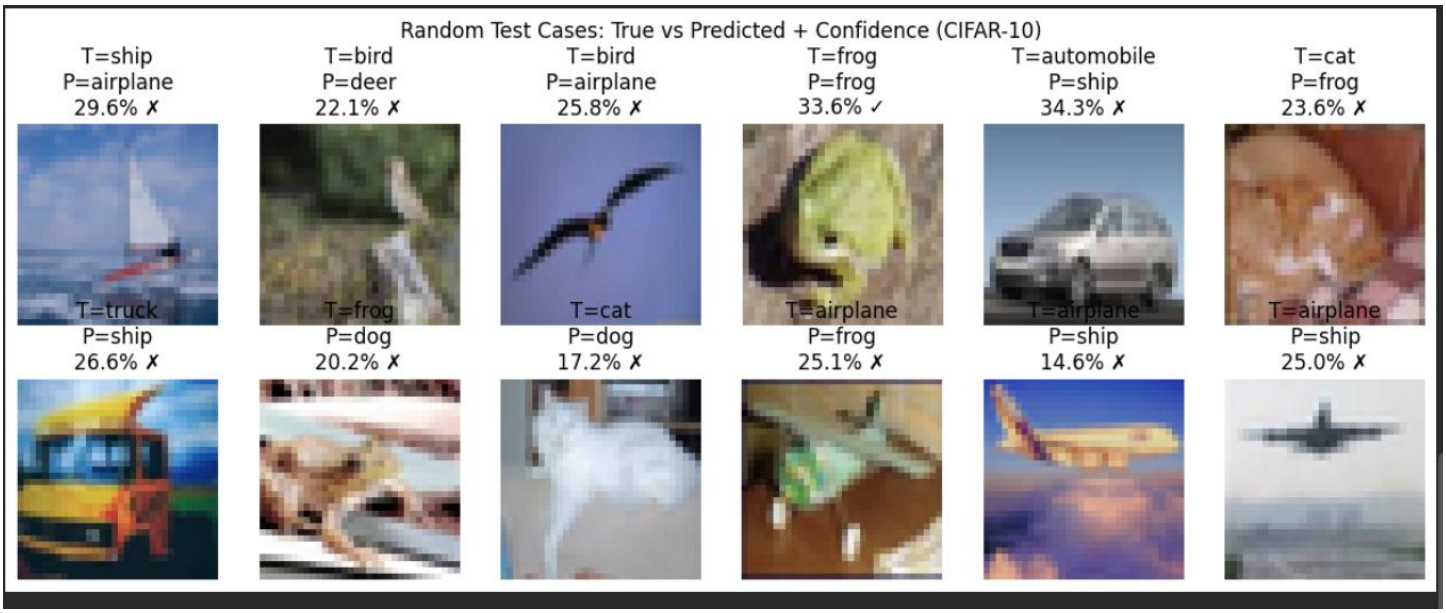
Task 3 :


Random Training Samples (CIFAR-10)
horse
ship
airplane
frog
automobile
frog
ship
airplane
frog
dog
bird
dog


Accuracy Curves (CIFAR-10, Dense-only)
Train Accuracy
Validation Accuracy
Accuracy
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
2
4
6
8
10
Epoch


Loss Curves (CIFAR-10, Dense-only)
Train Loss
Validation Loss
Loss
Epoch
2.10
2.05
2.00
1.95
1.90
1.85
1.80
2
4
6
8
10


Confusion Matrix (CIFAR-10 Test) - Colored
True Class
airplane 389 32 45 24 22 13 34 102 233 106
automobile 19 459 9 32 14 19 45 56 136 211
bird 98 21 105 67 192 59 221 164 41 32
cat 19 19 36 203 69 169 218 134 44 89
deer 32 8 56 64 315 31 263 158 29 44
dog 15 9 32 130 84 318 184 143 48 37
frog 6 18 23 81 109 43 604 63 8 45
horse 19 24 19 38 113 63 61 547 23 93
ship 111 58 7 36 5 34 11 31 553 154
truck 19 132 6 30 7 14 44 76 127 545
bird cat deer dog frog horse ship truck
600
500
400
300
200
100




```
Preprocessing:
- Normalized pixel values to [0, 1].
- Flattened each image from 32x32x3 to 3072 features.

Model (Dense-only):
- Dense(512, ReLU) + Dropout(0.4) + Dense(256, ReLU) + Dropout(0.3) + Dense(10, Softmax).

Training (10 epochs):
- Final Train Acc: 0.3284 | Final Val Acc: 0.3956
- Final Train Loss: 1.8299 | Final Val Loss: 1.7847

Testing:
- Test Loss: 1.7659 | Test Accuracy: 0.4038
```

```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

tf.random.set_seed(42)
np.random.seed(42)

# ----------------------------
# 1) Load CIFAR-10
# ----------------------------
(x_train_img, y_train), (x_test_img, y_test) = tf.keras.datasets.cifar10.load_data()
y_train = y_train.squeeze()  # (50000,1) -> (50000,)
y_test  = y_test.squeeze()   # (10000,1) -> (10000,)

class_names = [
    "airplane", "automobile", "bird", "cat", "deer",
    "dog", "frog", "horse", "ship", "truck"
]

print("=== DATASET OVERVIEW (CIFAR-10) ===")
print("Train images shape:", x_train_img.shape, " Train labels shape:", y_train.shape)
print("Test  images shape:", x_test_img.shape,  " Test  labels shape:", y_test.shape)
print("Pixel range (train):", (x_train_img.min(), x_train_img.max()))
print("Pixel range (test): ", (x_test_img.min(), x_test_img.max()))
print("Classes (0-9):", list(range(10)))
print("Class names:", class_names)

# ----------------------------
# 2) Explore dataset: distribution + sample images
# ----------------------------
def plot_class_distribution(labels, title):
    counts = np.bincount(labels, minlength=10)
    plt.figure(figsize=(10,3))
    plt.bar(range(10), counts)
    plt.xticks(range(10), class_names, rotation=20, ha="right")
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.title(title)
    plt.tight_layout()
    plt.show()
    return counts

train_counts = plot_class_distribution(y_train, "Train Class Distribution (CIFAR-10)")
test_counts  = plot_class_distribution(y_test,  "Test Class Distribution (CIFAR-10)")
print("Train class counts:", train_counts)
print("Test  class counts:", test_counts)

def show_samples(images, labels, n=12, title="Random Samples"):
    idx = np.random.choice(len(images), n, replace=False)
    cols = 6
    rows = int(np.ceil(n / cols))
    plt.figure(figsize=(12, 2.4*rows))
    for i, k in enumerate(idx):
        plt.subplot(rows, cols, i+1)
        plt.imshow(images[k])
        plt.title(class_names[labels[k]])
        plt.axis("off")
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

show_samples(x_train_img, y_train, n=12, title="Random Training Samples (CIFAR-10)")

# ----------------------------
# 3) Preprocess: normalize + flatten
# ----------------------------
x_train = (x_train_img.astype("float32") / 255.0).reshape(-1, 32*32*3)
x_test  = (x_test_img.astype("float32")  / 255.0).reshape(-1, 32*32*3)

# ----------------------------
# 4) Build Dense-only model
#    (CIFAR-10 is harder for dense-only; keep it small-ish but capable)
# ----------------------------
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(32*32*3,)),
    tf.keras.layers.Dense(512, activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(256, activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
```

```python
print("\n=== MODEL SUMMARY ===")
model.summary()

# ----------------------------
# 5) Train (10 epochs)
# ----------------------------
EPOCHS = 10
history = model.fit(
    x_train, y_train,
    epochs=EPOCHS,
    batch_size=128,
    validation_split=0.1,
    verbose=1
)

# ----------------------------
# 6) Plot training + validation accuracy curves (Task requirement)
# ----------------------------
def plot_accuracy_curves(hist):
    h = hist.history
    ep = range(1, len(h["accuracy"]) + 1)
    plt.figure(figsize=(7,4))
    plt.plot(ep, h["accuracy"], label="Train Accuracy")
    plt.plot(ep, h["val_accuracy"], label="Validation Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.title("Accuracy Curves (CIFAR-10, Dense-only)")
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_accuracy_curves(history)

# (Optional bonus: loss curves)
def plot_loss_curves(hist):
    h = hist.history
    ep = range(1, len(h["loss"]) + 1)
    plt.figure(figsize=(7,4))
    plt.plot(ep, h["loss"], label="Train Loss")
    plt.plot(ep, h["val_loss"], label="Validation Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.title("Loss Curves (CIFAR-10, Dense-only)")
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_loss_curves(history)

# ----------------------------
# 7) Test evaluation
# ----------------------------
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print("\n=== TEST PERFORMANCE ===")
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")

# ----------------------------
# 8) Colored Confusion Matrix + Most common confusions
# ----------------------------
all_probs = model.predict(x_test, verbose=0)
y_pred = np.argmax(all_probs, axis=1)

def confusion_matrix_np(y_true, y_pred, num_classes=10):
    cm = np.zeros((num_classes, num_classes), dtype=np.int32)
    for t, p in zip(y_true, y_pred):
        cm[t, p] += 1
    return cm

cm = confusion_matrix_np(y_test, y_pred, num_classes=10)

plt.figure(figsize=(8,6))
plt.imshow(cm, cmap="plasma")   # colored heatmap
plt.title("Confusion Matrix (CIFAR-10 Test) - Colored")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.xticks(range(10), class_names, rotation=25, ha="right")
plt.yticks(range(10), class_names)
plt.colorbar()

# write counts on cells
thresh = cm.max() * 0.6
for i in range(10):
    for j in range(10):
        plt.text(j, i, cm[i, j],
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black",
                 fontsize=8)
```

```
plt.tight_layout()
plt.show()

# show most frequent confusions
off_diag = cm.copy()
np.fill_diagonal(off_diag, 0)
pairs = []
for i in range(10):
    for j in range(10):
        if i != j and off_diag[i, j] > 0:
            pairs.append((off_diag[i, j], i, j))
pairs.sort(reverse=True)

print("\n=== MOST COMMON CONFUSIONS (true -> predicted) ===")
for c, t, p in pairs[:10]:
    print(f"- {class_names[t]} -> {class_names[p]}: {c} times")

# ----------------------------
# 9) Test cases (random predictions shown as images)
# ----------------------------
def predict_test_cases(x_test_images, y_true, y_pred, probs, n_cases=12):
    idx = np.random.choice(len(x_test_images), n_cases, replace=False)
    cols = 6
    rows = int(np.ceil(n_cases / cols))
    plt.figure(figsize=(12, 2.4*rows))
    for i, k in enumerate(idx):
        plt.subplot(rows, cols, i+1)
        plt.imshow(x_test_images[k])
        conf = np.max(probs[k])
        correct = (y_pred[k] == y_true[k])
        mark = "✓" if correct else "✗"
        plt.title(f"T={class_names[y_true[k]]}\nP={class_names[y_pred[k]]}\n{conf*100:.1f}% {mark}")
        plt.axis("off")
    plt.suptitle("Random Test Cases: True vs Predicted + Confidence (CIFAR-10)")
    plt.tight_layout()
    plt.show()

    print("\nSample Predictions (index, true, pred, confidence):")
    for k in idx:
        print(f"- idx={k:5d} | true={class_names[y_true[k]]:10s} | pred={class_names[y_pred[k]]:10s} | conf={np.max(probs[k]):.4f}")

predict_test_cases(x_test_img, y_test, y_pred, all_probs, n_cases=12)

# ----------------------------
# 10) Misclassified examples (visual)
# ----------------------------
def show_misclassified(x_test_images, y_true, y_pred, probs, n=12):
    wrong_idx = np.where(y_true != y_pred)[0]
    if len(wrong_idx) == 0:
        print("No misclassifications found (rare).")
        return
    pick = np.random.choice(wrong_idx, size=min(n, len(wrong_idx)), replace=False)

    cols = 6
    rows = int(np.ceil(len(pick) / cols))
    plt.figure(figsize=(12, 2.4*rows))
    for i, k in enumerate(pick):
        plt.subplot(rows, cols, i+1)
        plt.imshow(x_test_images[k])
        conf = np.max(probs[k])
        plt.title(f"T={class_names[y_true[k]]}\nP={class_names[y_pred[k]]}\n({conf*100:.1f}%)")
        plt.axis("off")
    plt.suptitle("Misclassified Test Examples (CIFAR-10)")
    plt.tight_layout()
    plt.show()

show_misclassified(x_test_img, y_test, y_pred, all_probs, n=12)

# ----------------------------
# 11) Small detailed report
# ----------------------------
train_acc_last = history.history["accuracy"][-1]
val_acc_last   = history.history["val_accuracy"][-1]
train_loss_last = history.history["loss"][-1]
val_loss_last   = history.history["val_loss"][-1]

print("\n" + "="*60)
print("SMALL DETAILED REPORT (Task 3: CIFAR-10)")
print("="*60)
print("Exploration:")
print("- Loaded CIFAR-10: 50,000 training images and 10,000 test images.")
print("- Each image is 32x32 with 3 color channels (RGB).")
print("- Displayed random sample images and plotted class distributions.")
print("\nPreprocessing:")
print("- Normalized pixel values to [0, 1].")
print("- Flattened each image from 32x32x3 to 3072 features.")
print("\nModel (Dense-only):")
```

```
print("- Dense(512, ReLU) + Dropout(0.4) + Dense(256, ReLU) + Dropout(0.3) + Dense(10, Softmax).")
print("\nTraining (10 epochs):")
print(f"- Final Train Acc: {train_acc_last:.4f} | Final Val Acc: {val_acc_last:.4f}")
print(f"- Final Train Loss: {train_loss_last:.4f} | Final Val Loss: {val_loss_last:.4f}")
print("\nTesting:")
print(f"- Test Loss: {test_loss:.4f} | Test Accuracy: {test_acc:.4f}")
print("\nNotes:")
print("- CIFAR-10 is harder than MNIST/Fashion-MNIST; Dense-only models usually perform worse than CNNs.")
print("- Confusion matrix and misclassified examples help identify which classes are most frequently mixed up.")
print("="*60)
```

Conclusion

In these three tasks, we built and tested fully connected (dense) neural networks on three different image classification datasets: MNIST, Fashion-MNIST, and CIFAR-10.
The goal was to understand how dense neural networks work, how data is prepared, and how model performance changes with dataset complexity.

Task 1: MNIST (Handwritten Digits)

- Loaded the MNIST dataset of handwritten digits (0–9).
- Normalized the pixel values and flattened each image.
- Built a small neural network using only dense layers.
- Trained the model for 10 epochs.
- Evaluated the model using accuracy, loss curves, confusion matrix, and test examples.
- The model achieved **very high accuracy**.
- Training and validation curves showed stable learning.
- The confusion matrix showed very few errors.
- This task proved that dense networks work **very well** for simple image data.

Conclusion

MNIST is an easy dataset, and a simple fully connected neural network is enough to classify digits accurately.

Task 2: Fashion-MNIST (Clothing Images)

- Explored clothing image classes (shirts, shoes, bags, etc.).
- Normalized and flattened the images.
- Built a dense neural network with dropout for regularization.
- Trained and tested the model.
- Visualized accuracy, loss curves, confusion matrix, and misclassified samples.
- Accuracy was lower than MNIST, but still reasonable.
- The confusion matrix showed that similar clothes (e.g., shirt vs coat) are often confused.
- Misclassified images helped us understand where the model struggles.

Conclusion

Fashion-MNIST is more complex than MNIST.Dense networks can still work, but they start to struggle with visual similarity between classes.

Task 3: CIFAR-10 (Color Object Images)

- Worked with color images (airplanes, cars, animals, etc.).
- Normalized and flattened RGB images.
- Built a larger dense network to handle higher complexity.
- Trained the model and plotted training and validation accuracy.
- Analyzed results using confusion matrix and test predictions.
- Accuracy was significantly lower than the previous tasks.
- The model confused similar objects (e.g., cats vs dogs, trucks vs cars).
- This showed the limitations of dense-only networks for complex image data.

Conclusion

CIFAR-10 is a hard dataset for fully connected networks.This task clearly shows why Convolutional Neural Networks (CNNs) are better for image classification.