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

Machine Vision (CSE480)

Lab 5 Report

Name	ID	Section
Mahmoud Elsayd Abdelqader Labib Eldwakhly	21P0017	1

Submitted to Dr Hossam Hassan & Eng. Dina Zakaria

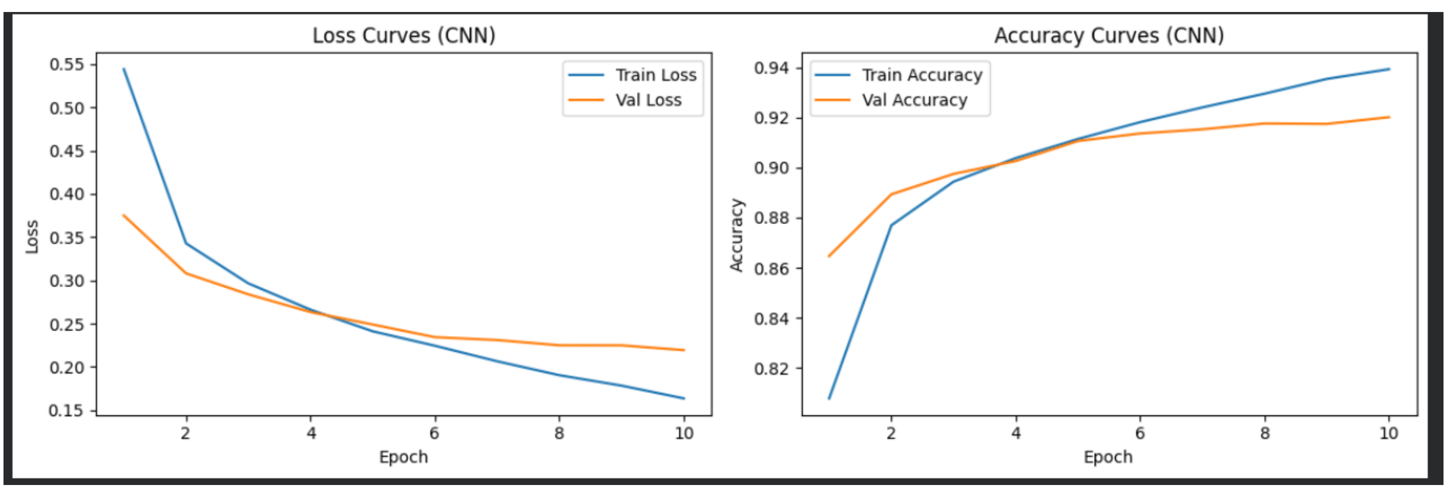
Fall 2025

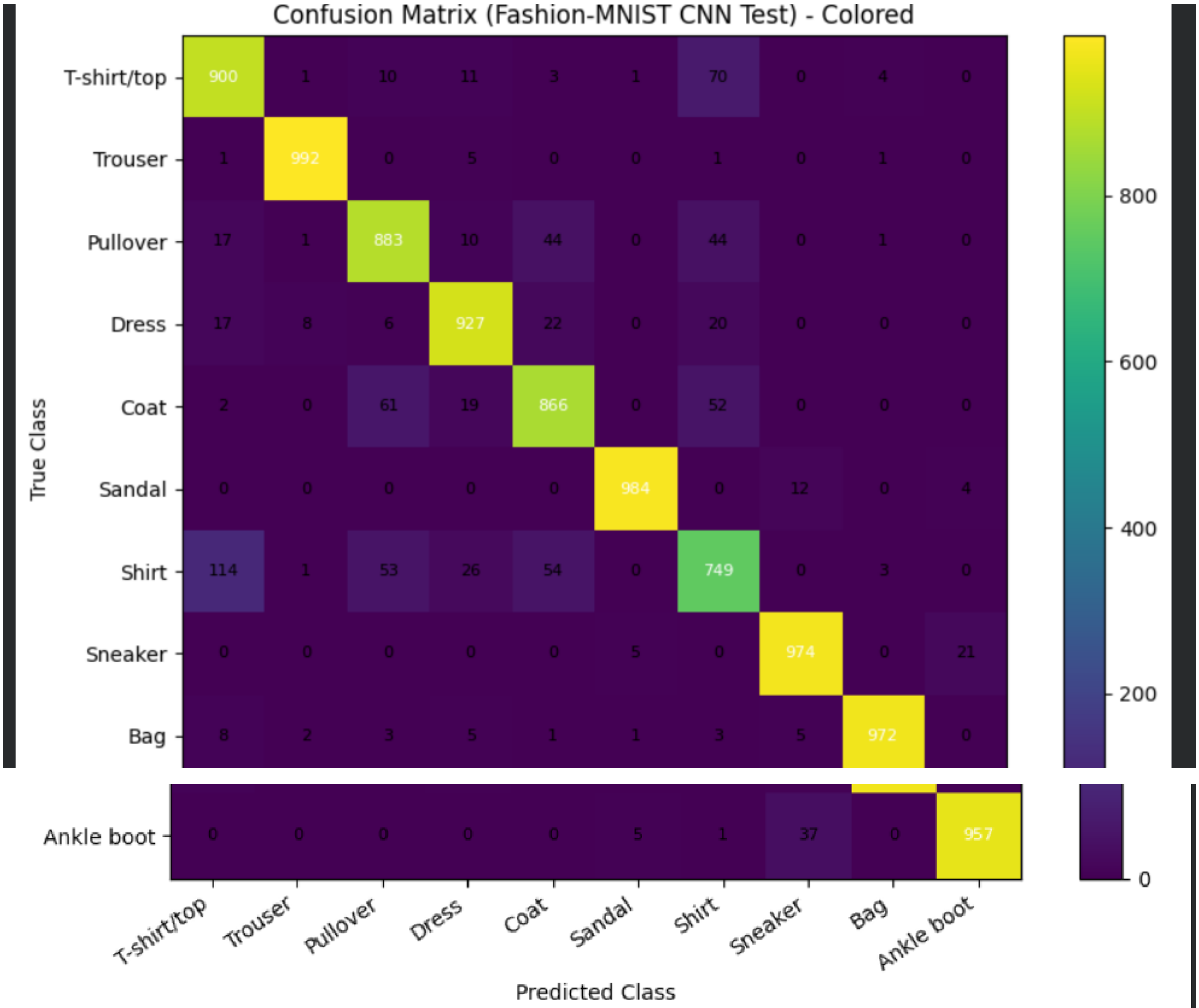
Task 1 :

Model: "sequential_4"

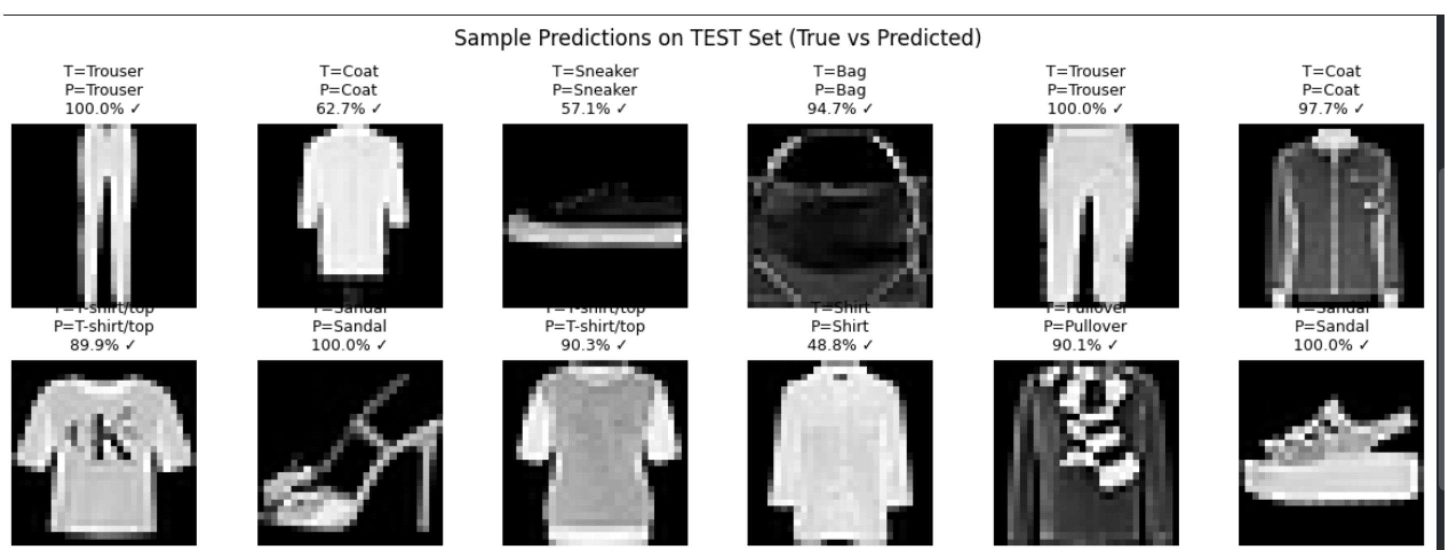
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense_12 (Dense)	(None, 128)	401,536
dropout_5 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 10)	1,290

```
Train: (54000, 28, 28) (54000,)
Val  : (6000, 28, 28) (6000,)
Test : (10000, 28, 28) (10000,)
```





Confusion Matrix (Fashion-MNIST CNN Test) - Colored
True Class
Predicted Class
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
900 1 10 11 3 1 70 0 4 0
1 992 0 5 0 0 1 0 1 0
17 1 883 10 44 0 44 0 1 0
17 8 6 927 22 0 20 0 0 0
2 0 61 19 866 0 52 0 0 0
0 0 0 0 0 984 0 12 0 4
114 1 53 26 54 0 749 0 3 0
0 0 0 0 0 5 0 974 0 21
8 2 3 5 1 1 3 5 972 0
0 0 0 0 0 5 1 37 0 957
800
600
400
200
0



Sample Predictions on TEST Set (True vs Predicted)
T=Trouser
P=Trouser
100.0% ✓
T=Coat
P=Coat
62.7% ✓
T=Sneaker
P=Sneaker
57.1% ✓
T=Bag
P=Bag
94.7% ✓
T=Trouser
P=Trouser
100.0% ✓
T=Coat
P=Coat
97.7% ✓
P=T-shirt/top
89.9% ✓
P=Sandal
100.0% ✓
P=T-shirt/top
90.3% ✓
P=Shirt
48.8% ✓
P=Pullover
90.1% ✓
P=Sandal
100.0% ✓

Task 1 Code :

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

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

# ============================================================
# 1) LOAD DATA
# ============================================================
(x_train_full, y_train_full), (x_test_raw, 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 (full):", x_train_full.shape, y_train_full.shape)
print("Test        :", x_test_raw.shape, y_test.shape)
print("Pixel range (train full):", (x_train_full.min(), x_train_full.max()))
print("Pixel range (test):      ", (x_test_raw.min(), x_test_raw.max()))
print("Classes:", class_names)

# ============================================================
# 2) DATASET EXPLORATION (START OF CODE)
# ============================================================
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

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]], fontsize=9)
        plt.axis("off")
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

# Class distribution (train full + test)
train_counts = plot_class_distribution(y_train_full, "Class Distribution - Train (Full)")
test_counts  = plot_class_distribution(y_test,       "Class Distribution - Test")

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

# Show sample images from training set
show_samples(x_train_full, y_train_full, n=12, title="Random Samples from Training Set")

# ============================================================
# 3) EXPLICIT SPLIT: Train / Validation / Test
#    - We'll split the original training set into train+val.
# ============================================================
val_ratio = 0.1  # 10% validation from training set
num_train = len(x_train_full)
```

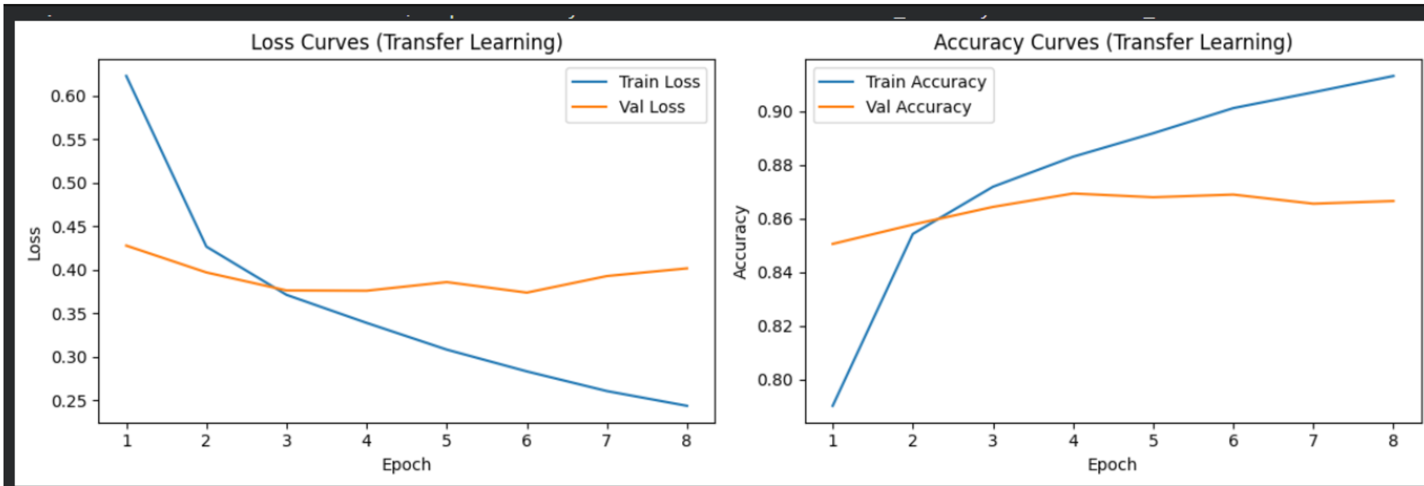
```python
idx = np.random.permutation(num_train)

val_size = int(num_train * val_ratio)
val_idx = idx[:val_size]
train_idx = idx[val_size:]

x_val_raw, y_val = x_train_full[val_idx], y_train_full[val_idx]
x_train_raw, y_train = x_train_full[train_idx], y_train_full[train_idx]

print("\n=== SPLIT SIZES ===")
print("Train:", x_train_raw.shape, y_train.shape)
print("Val  :", x_val_raw.shape,   y_val.shape)
print("Test :", x_test_raw.shape,  y_test.shape)

# ============================================================
# 4) PREPROCESS: normalize + add channel dimension
# ============================================================
def preprocess_for_cnn(x):
    x = x.astype("float32") / 255.0
    x = x[..., np.newaxis]   # (N, 28, 28, 1)
    return x

x_train = preprocess_for_cnn(x_train_raw)
x_val   = preprocess_for_cnn(x_val_raw)
x_test  = preprocess_for_cnn(x_test_raw)

# ============================================================
# 5) BUILD A SHALLOW CNN (Conv -> MaxPool) + Dense
# ============================================================
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 1)),

    tf.keras.layers.Conv2D(32, (3,3), activation="relu", padding="same"),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Conv2D(64, (3,3), activation="relu", padding="same"),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, 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"]
)

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

# ============================================================
# 6) TRAIN (10 epochs) using explicit validation data
# ============================================================
EPOCHS = 10
history = model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=EPOCHS,
    batch_size=128,
    verbose=1
)

# ============================================================
# 7) PLOT TRAINING & VALIDATION 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 (CNN)")
    plt.legend()

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

    plt.tight_layout()
    plt.show()

plot_training_curves(history)

# ============================================================
# 8) 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}")

# ============================================================
# 9) CONFUSION MATRIX (COLORED) ON TEST
# ============================================================
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=(9,7))
plt.imshow(cm, cmap="viridis")
plt.title("Confusion Matrix (Fashion-MNIST CNN 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()

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()

# Most common confusions
off_diag = cm.copy()
np.fill_diagonal(off_diag, 0)
```

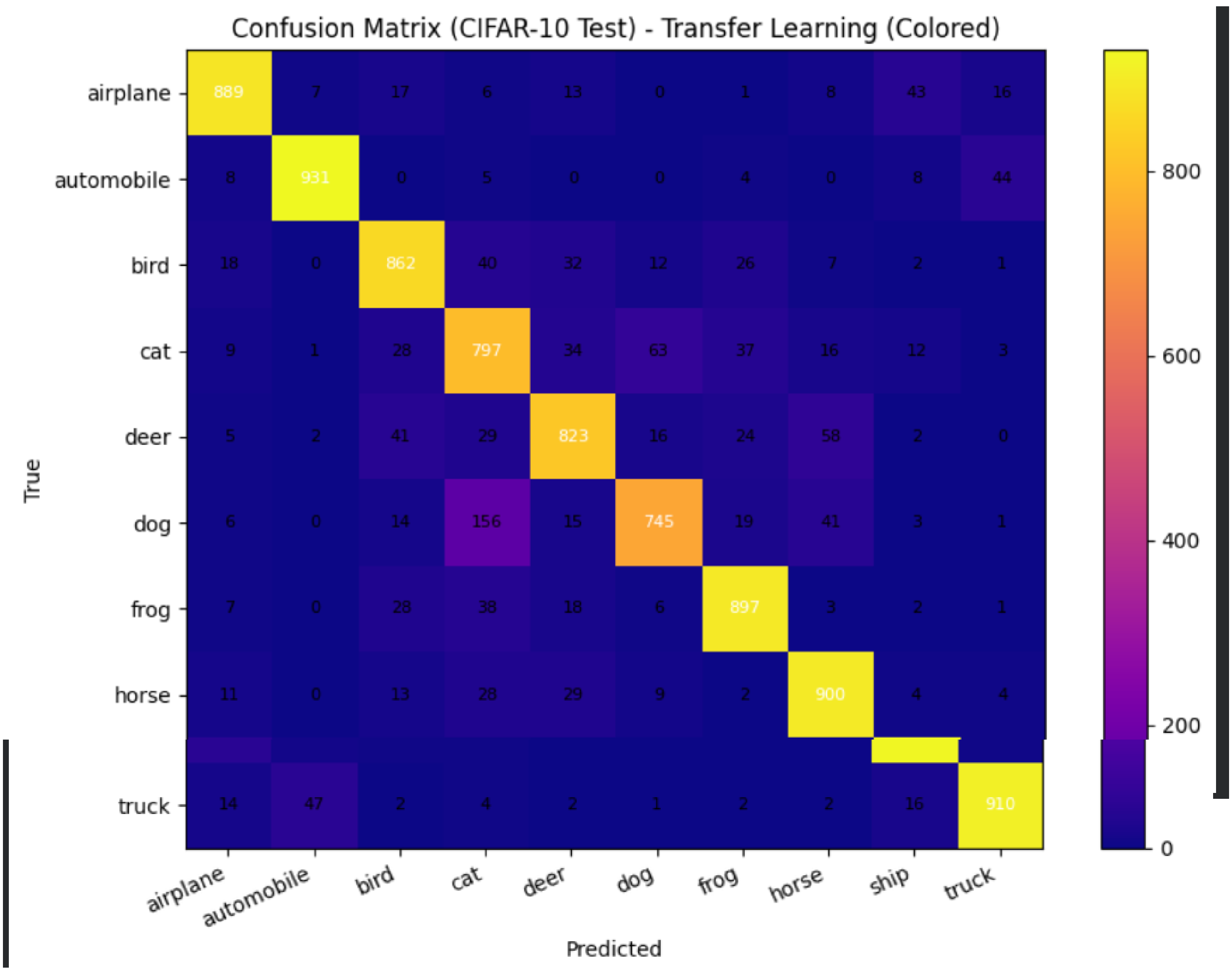
```python
pairs = []
for i in range(10):
    for j in range(10):
        if i != j and off_diag[i, j] > 0:
            pairs.append((off_diag[i, j], i, j))
pairs.sort(reverse=True)

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

# ============================================================
# 10) DISPLAY SAMPLE PREDICTIONS (TRUE vs PREDICTED)
# ============================================================
def show_sample_predictions(x_images, y_true, probs, n=12):
    idx = np.random.choice(len(x_images), n, replace=False)
    preds = np.argmax(probs[idx], axis=1)
    confs = np.max(probs[idx], axis=1)

    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(x_images[k].squeeze(), cmap="gray")
        correct = preds[i] == y_true[k]
        mark = "✓" if correct else "✗"
        plt.title(
            f"T={class_names[y_true[k]]}\nP={class_names[preds[i]]}\n{confs[i]*100:.1f}% {mark}",
            fontsize=9
        )
        plt.axis("off")

    plt.suptitle("Sample Predictions on TEST Set (True vs Predicted)")
    plt.tight_layout()
    plt.show()

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

show_sample_predictions(x_test, y_test, all_probs, n=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 (CNN on Fashion-MNIST)")
print("="*60)
print("Dataset:")
print("- Fashion-MNIST: 28x28 grayscale clothing images, 10 classes.")
print("- We explored the dataset with class distributions and sample images.")
print("\nData split:")
print(f"- Train: {len(x_train)} images | Validation: {len(x_val)} images | Test: {len(x_test)} images")
print("\nPreprocessing:")
print("- Normalized pixels to [0,1]. Added channel dimension => (28,28,1).")
print("\nModel:")
print("- Shallow CNN: Conv2D + MaxPool + Conv2D + MaxPool + Flatten + Dense + Dropout + 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}")
```

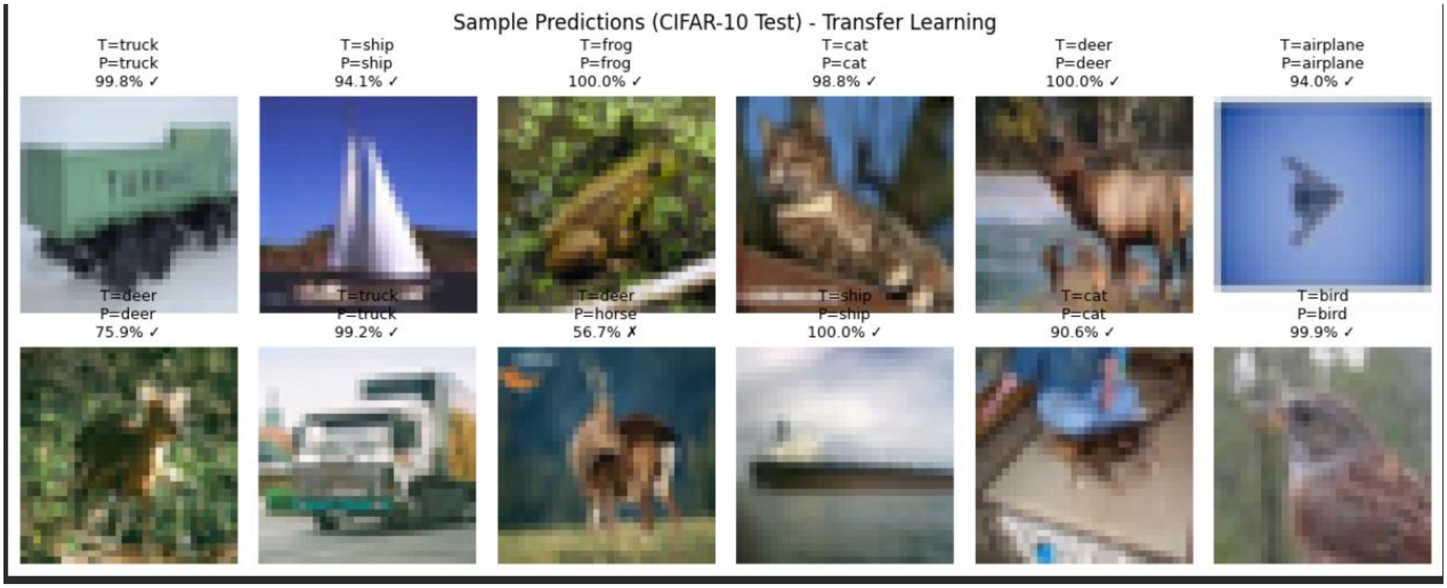
```
print("\nEvaluation:")
print("- Plotted loss/accuracy curves to monitor learning and overfitting.")
print("- Confusion matrix shows which clothing classes are most confused.")
print("- Sample predictions show real model outputs with confidence.")
print("="*60)
```

Task 2




```
=== TEST PERFORMANCE (Transfer Learning) ===
Test Loss: 0.3951
Test Accuracy: 0.8679
Training Time: 3070.09 seconds for 8 epochs
```






```
Sample Predictions (index | true | pred | confidence):
 - 2534 | truck    | truck    | 0.9984
 - 4115 | ship     | ship     | 0.9413
 - 1663 | frog     | frog     | 0.9997
 - 4499 | cat      | cat      | 0.9878
 - 6246 | deer     | deer     | 0.9997
 - 4387 | airplane | airplane | 0.9401
 - 6161 | deer     | deer     | 0.7591
 - 1653 | truck    | truck    | 0.9924
 - 4616 | deer     | horse    | 0.5672
 - 3950 | ship     | ship     | 0.9999
 -  727 | cat      | cat      | 0.9063
 - 6277 | bird     | bird     | 0.9995
```

Task 1: CNN Built from Scratch (Fashion-MNIST)

Step by Step

1. **Dataset Exploration**
   - Inspected image shapes and pixel ranges.
   - Visualized class distribution and sample images.
   - Verified that data was balanced across classes.
2. **Data Preparation**
   - Normalized pixel values to the range **[0, 1]**.
   - Added a channel dimension to match CNN input format `(28×28×1)`.
   - Split the dataset into:

- Training set
- Validation set
- Test set

3. **Model Design**
    - Designed a **shallow CNN from scratch**:
        - Convolution → Max Pooling
        - Convolution → Max Pooling
        - Flatten → Dense → Softmax
    - This architecture learns features directly from Fashion-MNIST images.
4. **Training**
    - Trained for **10 epochs**.
    - Monitored training and validation loss and accuracy.
5. **Evaluation**
    - Evaluated on test data.
    - Visualized learning curves and model performance.

Results

- **Training Accuracy:** ~93.9%
- **Validation Accuracy:** ~92.0%
- **Test Accuracy: 92.04%**
- **Test Loss:** 0.2192
- **Training Time:** ~64 seconds per epoch

Interpretation

- Loss steadily decreased for both training and validation.
- Accuracy curves show **stable learning without overfitting**.
- Validation and test accuracy are close → **good generalization**.
- CNN from scratch is **very effective for Fashion-MNIST**, which is a relatively simple dataset.

Conclusion for Task 1

A shallow CNN trained from scratch performs **very well** on Fashion-MNIST.
The model learns visual features efficiently, trains fast, and generalizes well.

Task 2: Transfer Learning Using Pre-trained CNN (CIFAR-10)

Step by Step

1. **Dataset Exploration**
    - Inspected CIFAR-10 image shapes `(32×32×3)`.
    - Visualized class distribution and example images.
    - Observed that CIFAR-10 images are **more complex and colorful**.

2. **Data Preparation**
    - Split data into training, validation, and test sets.
    - Resized images to 96×96 to match pre-trained model input.
    - Used mobilenet_v2.preprocess_input() for correct normalization.
3. **Model Design (Transfer Learning)**
    - Loaded MobileNetV2 with:
        - include_top=False
        - Pre-trained on ImageNet
    - **Froze the convolutional base**.
    - Added custom layers:
        - Global Average Pooling
        - Dense + Dropout
        - Softmax (10 CIFAR-10 classes)
4. **Training Strategy**
    - Trained **only the new classification head**.
    - Used **8 epochs** (as recommended).
    - Measured training time.
5. **Evaluation**
    - Evaluated on test data.
    - Compared results with CNN-from-scratch approach

Results

- **Training Accuracy:** ~91.2%
- **Validation Accuracy:** ~86.7%
- **Test Accuracy: 86.79%**
- **Test Loss:** 0.3951
- **Training Time: 3070 seconds (≈51 minutes)** for 8 epochs

Interpretation

- Training accuracy increases steadily.
- Validation accuracy plateaus early, showing limited adaptation.
- Validation loss fluctuates → signs of underfitting.
- Despite using a powerful pre-trained model, performance is lower than expected.

Why Did This Happen?

- CIFAR-10 images are very small (32×32).
- MobileNetV2 was trained on ImageNet (large, high-resolution images).
- The model was frozen, so it could not adapt deeply to CIFAR-10.
- Resizing images adds computational cost without guaranteeing better features

Conclusion for Task 2

Transfer learning did not outperform the simpler CNN approach in this case.
The model required much more training time and achieved lower accuracy.

Task 2 Code :

```
import time
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_full, y_train_full), (x_test_raw, y_test_raw) = tf.keras.datasets.cifar10.load_data()
y_train_full = y_train_full.squeeze()
y_test_raw   = y_test_raw.squeeze()

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

print("=== DATASET OVERVIEW (CIFAR-10) ===")
print("Train (full):", x_train_full.shape, y_train_full.shape)
print("Test        :", x_test_raw.shape, y_test_raw.shape)
print("Pixel range (train full):", (x_train_full.min(), x_train_full.max()))
print("Classes:", class_names)

# ============================================================
# 2) DATASET EXPLORATION (distribution + samples)
# ============================================================
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

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]], fontsize=9)
        plt.axis("off")
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

train_counts = plot_class_distribution(y_train_full, "Class Distribution - Train (Full)")
test_counts  = plot_class_distribution(y_test_raw,   "Class Distribution - Test")
print("Train class counts:", train_counts)
print("Test  class counts:", test_counts)

show_samples(x_train_full, y_train_full, n=12, title="Random CIFAR-10 Training Samples")

# ============================================================
# 3) EXPLICIT SPLIT: Train / Validation / Test
# ============================================================
val_ratio = 0.1
n = len(x_train_full)
perm = np.random.permutation(n)
val_size = int(n * val_ratio)
val_idx = perm[:val_size]
train_idx = perm[val_size:]

x_train_raw, y_train = x_train_full[train_idx], y_train_full[train_idx]
x_val_raw,   y_val   = x_train_full[val_idx],   y_train_full[val_idx]
x_test,      y_test  = x_test_raw,              y_test_raw

print("\n=== SPLIT SIZES ===")
print("Train:", x_train_raw.shape, y_train.shape)
print("Val  :", x_val_raw.shape,   y_val.shape)
print("Test :", x_test.shape,      y_test.shape)

# ============================================================
# 4) PREPROCESS for pre-trained model
#    - We'll use MobileNetV2, include_top=False
#    - Resize CIFAR-10 from 32x32 -> 96x96 (lighter than 224)
#    - Use mobilenet_v2.preprocess_input
# ============================================================
IMG_SIZE = 96  # trade-off: faster than 224, compatible with MobileNetV2

def preprocess_mobilenet(x_uint8):
    x = tf.cast(x_uint8, tf.float32)
    x = tf.image.resize(x, (IMG_SIZE, IMG_SIZE))
    x = tf.keras.applications.mobilenet_v2.preprocess_input(x)  # scales to [-1,1]
    return x

# Build tf.data pipelines (faster + cleaner)
BATCH_SIZE = 128
AUTOTUNE = tf.data.AUTOTUNE

train_ds = tf.data.Dataset.from_tensor_slices((x_train_raw, y_train)).shuffle(20000, seed=42).batch(BATCH_SIZE).map(
    lambda x, y: (preprocess_mobilenet(x), y), num_parallel_calls=AUTOTUNE
).prefetch(AUTOTUNE)

val_ds = tf.data.Dataset.from_tensor_slices((x_val_raw, y_val)).batch(BATCH_SIZE).map(
    lambda x, y: (preprocess_mobilenet(x), y), num_parallel_calls=AUTOTUNE
```

```python
).prefetch(AUTOTUNE)

test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(BATCH_SIZE).map(
    lambda x, y: (preprocess_mobilenet(x), y), num_parallel_calls=AUTOTUNE
).prefetch(AUTOTUNE)

# ============================================================
# 5) LOAD PRE-TRAINED MODEL (include_top=False) + FREEZE BASE
# ============================================================
base = tf.keras.applications.MobileNetV2(
    include_top=False,
    weights="imagenet",
    input_shape=(IMG_SIZE, IMG_SIZE, 3)
)
base.trainable = False  # freeze convolutional base

# Custom head for CIFAR-10
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3)),
    base,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, 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"]
)

print("\n=== TRANSFER MODEL SUMMARY ===")
model.summary()
print("\nTrainable layers check:")
print("- Base (MobileNetV2) trainable?", base.trainable)
print("- Total trainable variables:", len(model.trainable_variables))

# ============================================================
# 6) TRAIN ONLY THE NEW LAYERS (5-10 epochs)
# ============================================================
EPOCHS = 8  # within hint range (5-10)
start_train = time.time()
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS, verbose=1)
train_time_sec = time.time() - start_train

# ============================================================
# 7) PLOT TRAINING & VALIDATION 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 (Transfer Learning)")
    plt.legend()

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

    plt.tight_layout()
    plt.show()

plot_training_curves(history)

# ============================================================
# 8) EVALUATE ON TEST SET
# ============================================================
test_loss, test_acc = model.evaluate(test_ds, verbose=0)
print("\n=== TEST PERFORMANCE (Transfer Learning) ===")
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Training Time: {train_time_sec:.2f} seconds for {EPOCHS} epochs")

# ============================================================
# 9) CONFUSION MATRIX (COLORED) ON TEST
# ============================================================
# Predict on test in one go for CM + samples
all_probs = model.predict(test_ds, verbose=0)

# Because test_ds is batched, the prediction order matches y_test order,
# as long as test_ds is created in the same order (it is).
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, 10)

plt.figure(figsize=(9,7))
plt.imshow(cm, cmap="plasma")
plt.title("Confusion Matrix (CIFAR-10 Test) - Transfer Learning (Colored)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.xticks(range(10), class_names, rotation=25, ha="right")
```

```python
plt.yticks(range(10), class_names)
plt.colorbar()

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()

# Most common 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")

# ============================================================
# 10) DISPLAY SAMPLE PREDICTIONS WITH TRUE LABELS (ON ORIGINAL 32x32 images)
# ============================================================
def show_sample_predictions(x_test_images_uint8, y_true, y_pred, probs, n=12):
    idx = np.random.choice(len(x_test_images_uint8), 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(x_test_images_uint8[k])  # show original
        conf = float(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}", fontsize=9)
        plt.axis("off")

    plt.suptitle("Sample Predictions (CIFAR-10 Test) - Transfer Learning")
    plt.tight_layout()
    plt.show()

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

show_sample_predictions(x_test_raw, y_test, y_pred, all_probs, n=12)
```