APPENDIX

## 1. Data Loading and Preparation

Load the CIFAR-10 dataset and normalise pixel values.

```
# !pip install -q tensorflow pandas scikit-learn


import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split


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


# Class names for CIFAR-10
CLASS_NAMES = np.array([
    "airplane","automobile","bird","cat","deer",
    "dog","frog","horse","ship","truck"
])


# Load CIFAR-10
(x_train_full, y_train_full), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
y_train_full = y_train_full.flatten()
y_test       = y_test.flatten()


print(x_train_full.shape, y_train_full.shape)  # (50000, 32, 32, 3) (50000,)
print(x_test.shape, y_test.shape)              # (10000, 32, 32, 3) (10000,)
```

## 2. Dataset Partitioning

To show data split (Train/Validation/Test) and justify it.

```
# Stratified split to keep class balance
x_train, x_val, y_train, y_val = train_test_split(
    x_train_full, y_train_full,
    test_size=0.20, stratify=y_train_full, random_state=SEED
)

print("Train:", x_train.shape, y_train.shape)
print("Val  :", x_val.shape,   y_val.shape)
print("Test :", x_test.shape,  y_test.shape)
```

## 2B. Dataset Splitting and Validation

```
# Import libraries
from sklearn.model_selection import train_test_split
from tensorflow.keras.datasets import cifar10
import numpy as np

# Load CIFAR-10 dataset (50,000 train, 10,000 test)
(x_train_full, y_train_full), (x_test, y_test) = cifar10.load_data()

# Create Validation Set (20% of training data)
x_train, x_val, y_train, y_val = train_test_split(
    x_train_full,
    y_train_full,
    test_size=0.2,        # 20% for validation
    stratify=y_train_full, # Keeps class balance
    random_state=42        # Reproducibility
)

# Check dataset sizes
print("Training set:", x_train.shape, "Labels:", y_train.shape)
print("Validation set:", x_val.shape, "Labels:", y_val.shape)
print("Test set:", x_test.shape, "Labels:", y_test.shape)

# Confirm number of samples per class (optional check)
unique, counts = np.unique(y_train, return_counts=True)
print("\nTraining Class Distribution:")
for u, c in zip(unique, counts):
    print(f"Class {u}: {c} images")

unique_val, counts_val = np.unique(y_val, return_counts=True)
print("\nValidation Class Distribution:")
for u, c in zip(unique_val, counts_val):
    print(f"Class {u}: {c} images")
```

## 2C. Visual: Pie chart showing split ratio

```
import matplotlib.pyplot as plt

# Define your split data
split_labels = ['Training (40,000)', 'Validation (10,000)', 'Test (10,000)']
split_sizes = [40000, 10000, 10000]
colors = ['#66b3ff', '#99ff99', '#ffcc99']

# Create pie chart
```

```
plt.figure(figsize=(6, 6))
plt.pie(
    split_sizes,
    labels=split_labels,
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    wedgeprops={'edgecolor': 'black'}
)

plt.title('CIFAR-10 Dataset Partitioning', fontsize=14, fontweight='bold')
plt.show()
```

## 3. Dataset Meta Data Summary

```
def split_metadata(X, y_int, name):
    # overall stats
    mean = X.mean(axis=(0,1,2))
    std  = X.std(axis=(0,1,2))
    mn, mx = X.min(), X.max()

    # Ensure y_int is 1D before bincount
    y_int_flat = y_int.flatten()

    counts = np.bincount(y_int_flat, minlength=10)

    # build a tidy summary row
    row = {
        "split": name,
        "num_images": X.shape[0],
        "height": X.shape[1],
        "width": X.shape[2],
        "channels": X.shape[3],
        "dtype": str(X.dtype),
        "pixel_min": float(mn),
        "pixel_max": float(mx),
        "pixel_mean_R": float(mean[0]),
        "pixel_mean_G": float(mean[1]),
        "pixel_mean_B": float(mean[2]),
        "pixel_std_R":  float(std[0]),
        "pixel_std_G":  float(std[1]),
        "pixel_std_B":  float(std[2]),
    }
    # add class counts named by label
    for i, cname in enumerate(CLASS_NAMES):
        row[f"class_{i}_{cname}"] = int(counts[i])
    return row

meta = pd.DataFrame([
    split_metadata(x_train, y_train, "train"),
    split_metadata(x_val,   y_val,   "val"),
    split_metadata(x_test,  y_test,  "test"),
])

pd.set_option("display.max_columns", 200)
print(meta)

meta.to_csv("cifar10_split_metadata.csv", index=False)
```

## 3B. Visual Representation of Metadata Set

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

classes = ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
train_counts = [4000]*10
val_counts = [1000]*10
test_counts = [1000]*10

x = np.arange(len(classes))
width = 0.25

plt.figure(figsize=(10,5))
plt.bar(x - width, train_counts, width, label='Train', color='#66b3ff')
plt.bar(x, val_counts, width, label='Validation', color='#99ff99')
plt.bar(x + width, test_counts, width, label='Test', color='#ffcc99')

plt.xlabel('Classes', fontsize=12)
plt.ylabel('Number of Images', fontsize=12)
plt.title('Class Distribution Across Dataset Splits', fontsize=14, fontweight='bold')
plt.xticks(x, classes, rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

## 4. Normalisation

For Pixel Scaling

```
import matplotlib.pyplot as plt

# Select random sample indices
sample_indices = np.random.choice(len(x_train_full), size=5, replace=False)

# Original (unnormalized) images from x_train_full
original_imgs = x_train_full[sample_indices]
normalized_imgs = x_train_full[sample_indices].astype("float32") / 255.0

# Create figure
plt.figure(figsize=(12, 4))
for i in range(5):
    # Original image
    plt.subplot(2, 5, i+1)
    plt.imshow(original_imgs[i])
    plt.title("Original")
    plt.axis("off")

    # Normalized image
    plt.subplot(2, 5, i+6)
    plt.imshow(normalized_imgs[i])
    plt.title("Normalized")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

## 4B. Pixel Intensity Distribution Before and After Normalisation

```
import matplotlib.pyplot as plt

# Choose one random image
idx = np.random.randint(0, len(x_train_full))
img_original = x_train_full[idx]
img_normalized = img_original.astype("float32") / 255.0

# Plot RGB histograms before and after normalization
colors = ('r', 'g', 'b')
plt.figure(figsize=(12, 5))

# --- Before Normalization ---
plt.subplot(1, 2, 1)
for i, col in enumerate(colors):
    plt.hist(img_original[:, :, i].ravel(), bins=256, color=col, alpha=0.5)
plt.title("Before Normalization (Pixel Range: 0–255)")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.xlim(0, 255)

# --- After Normalization ---
plt.subplot(1, 2, 2)
for i, col in enumerate(colors):
    plt.hist(img_normalized[:, :, i].ravel(), bins=256, color=col, alpha=0.5)
plt.title("After Normalization (Pixel Range: 0–1)")
plt.xlabel("Normalized Pixel Value")
plt.ylabel("Frequency")
plt.xlim(0, 1)

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

## 5. One-HOT Encoding

```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.utils import to_categorical
from keras.datasets import cifar10

# Load dataset
(x_train, y_train), (_, _) = cifar10.load_data()

# One-hot encode labels
y_train_encoded = to_categorical(y_train, 10)

# Class names for CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Select a few random samples
num_samples = 5
indices = np.random.choice(len(x_train), num_samples, replace=False)
```

```
plt.figure(figsize=(14, 4))

for i, idx in enumerate(indices):
    img = x_train[idx]
    label = int(y_train[idx])
    one_hot = y_train_encoded[idx]

    plt.subplot(2, num_samples, i + 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(f"{class_names[label]}\nLabel: {label}", fontsize=10, weight='bold')

    plt.subplot(2, num_samples, i + 1 + num_samples)
    plt.imshow(one_hot.reshape(1, 10), cmap='Greens', aspect='auto')
    plt.xticks(np.arange(10), np.arange(10))
    plt.yticks([])
    plt.xlabel('Class Index', fontsize=8)
    plt.tight_layout()

plt.suptitle("One-Hot Encoding of CIFAR-10 Labels", fontsize=14, weight='bold')
plt.show()
```

## 6. Data Augumentation

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# --- 1) Make sure training images for preview are float32 in [0,1]
# (If you've already normalized to [0,1], this keeps them; if not, it rescales.)
x_train_vis = x_train.astype("float32")
if x_train_vis.max() > 1.5:
    x_train_vis /= 255.0

# --- 2) Define augmentation (training only). No rescale here because we already ensured
[0,1].
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    fill_mode="nearest"
)

# --- 3) Take a small batch and generate augmented versions (same order)
n = 6  # how many to show
samples = x_train_vis[:n]
aug_iter = datagen.flow(samples, batch_size=n, shuffle=False)
augmented = next(aug_iter)

# --- 4) Plot originals (top) vs augmented (bottom)
plt.figure(figsize=(12, 5))
for i in range(n):
    # original
    plt.subplot(2, n, i+1)
    plt.imshow(np.clip(samples[i], 0, 1))
    plt.axis("off")
    if i == 0:
        plt.ylabel("Original", fontsize=11, weight="bold")

    # augmented
    plt.subplot(2, n, n+i+1)
    plt.imshow(np.clip(augmented[i], 0, 1))
    plt.axis("off")
    if i == 0:
        plt.ylabel("Augmented", fontsize=11, weight="bold")

plt.suptitle("Sample Augmented CIFAR-10 Images", fontsize=14, weight="bold")
plt.tight_layout()
plt.show()
```

## 7. 🔧 Model 1 — Baseline CNN (no BN/Dropout, no augmentation)

Train the CNN model and monitor accuracy and loss across epochs.

```
# --- Setup
import numpy as np, tensorflow as tf, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns  # only for prettier confusion matrix

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

# --- Load CIFAR-10
```

```
(x_train_full, y_train_full), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
y_train_full = y_train_full.flatten()
y_test       = y_test.flatten()

# --- Stratified validation split (20% of train)
x_train, x_val, y_train, y_val = train_test_split(
    x_train_full, y_train_full, test_size=0.20, stratify=y_train_full, random_state=SEED
)

# --- Normalise to [0,1] and one-hot encode
x_train = x_train.astype("float32")/255.0
x_val   = x_val.astype("float32")/255.0
x_test  = x_test.astype("float32")/255.0

num_classes = 10
y_train_oh = tf.keras.utils.to_categorical(y_train, num_classes)
y_val_oh   = tf.keras.utils.to_categorical(y_val,   num_classes)
y_test_oh  = tf.keras.utils.to_categorical(y_test,  num_classes)

CLASS_NAMES = np.array(["airplane","automobile","bird","cat","deer",
                        "dog","frog","horse","ship","truck"])
```

## 8. Define the Baseline CNN

```
from tensorflow.keras import layers, models

def build_model1():
    model = models.Sequential([
        layers.Input(shape=(32,32,3)),
        layers.Conv2D(32, (3,3), activation='relu', padding='same'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, (3,3), activation='relu', padding='same'),
        layers.MaxPooling2D(),
        layers.Conv2D(128,(3,3), activation='relu', padding='same'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model

model1 = build_model1()
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
               loss='categorical_crossentropy',
               metrics=['accuracy'])

model1.summary()
```

## 9. Model Training (with simple callbacks)

```
callbacks = [
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                                         patience=3, verbose=1, min_lr=1e-5),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6,
                                     restore_best_weights=True, verbose=1)
]

EPOCHS = 15  # baseline: keep modest
BATCH  = 64
```

```
history1 = model1.fit(
    x_train, y_train_oh,
    validation_data=(x_val, y_val_oh),
    epochs=EPOCHS, batch_size=BATCH,
    callbacks=callbacks, verbose=1
)
```

## 10. Plot training/validation curves (Accuracy & Loss)

```
def plot_history(h, title_prefix="Model 1"):
    plt.figure(figsize=(10,4))
    # accuracy
    plt.subplot(1,2,1)
    plt.plot(h.history['accuracy'], label='train')
    plt.plot(h.history['val_accuracy'], label='val')
    plt.title(f"{title_prefix} Accuracy")
    plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend()
    # loss
    plt.subplot(1,2,2)
    plt.plot(h.history['loss'], label='train')
    plt.plot(h.history['val_loss'], label='val')
    plt.title(f"{title_prefix} Loss")
    plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()
    plt.tight_layout(); plt.show()

plot_history(history1, "Model 1 (Baseline)")
```

## 11. Evaluate on Train / Validation / Test

```
tr_loss, tr_acc = model1.evaluate(x_train, y_train_oh, verbose=0)
va_loss, va_acc = model1.evaluate(x_val,   y_val_oh,   verbose=0)
te_loss, te_acc = model1.evaluate(x_test,  y_test_oh,  verbose=0)

print(f"Model 1 — Accuracy  | Train: {tr_acc:.4f}  Val: {va_acc:.4f}  Test: {te_acc:.4f}")
print(f"Model 1 — Loss      | Train: {tr_loss:.4f}  Val: {va_loss:.4f}  Test: {te_loss:.4f}")
```

## 12. Confusion Matrix + Classification Report (on Test)

```
# Predict class indices on test set
y_pred_probs = model1.predict(x_test, batch_size=256, verbose=0)
y_pred = y_pred_probs.argmax(axis=1)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=False, cmap='Blues', fmt='d',
            xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Model 1 — Confusion Matrix (Test)")
plt.tight_layout(); plt.show()

# Precision / Recall / F1
print("Model 1 — Classification Report (Test):")
print(classification_report(y_test, y_pred, target_names=CLASS_NAMES))
```

## 13. Model 2 - Enhanced CNN (BN + Dropout + Augmentation)

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

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Reuse: x_train, x_val, x_test are float32 in [0,1]; y_*_oh are one-hot; CLASS_NAMES
defined earlier.

# --- Data augmentation as Keras preprocessing layers (train-time only) ---
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal", seed=42),
    tf.keras.layers.RandomRotation(0.1, seed=42),
    tf.keras.layers.RandomZoom(0.1, seed=42),
], name="augmentation")

from tensorflow.keras import layers, models

def build_model2():
    inputs = layers.Input(shape=(32,32,3))
    x = data_augmentation(inputs)             # applies only during training
    # Block 1
    x = layers.Conv2D(32, (3,3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv2D(32, (3,3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Dropout(0.25)(x)
    # Block 2
    x = layers.Conv2D(64, (3,3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv2D(64, (3,3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Dropout(0.25)(x)
    # Block 3
    x = layers.Conv2D(128, (3,3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Dropout(0.25)(x)
    # Head
    x = layers.Flatten()(x)
    x = layers.Dense(256)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(10, activation='softmax')(x)
    model = models.Model(inputs, outputs, name="Model2_EnhancedCNN")
    return model

model2 = build_model2()
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model2.summary()
```

```
# Train (longer with same safety callbacks)

callbacks2 = [
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                                         patience=3, verbose=1, min_lr=1e-5),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6,
                                     restore_best_weights=True, verbose=1)
]

EPOCHS2 = 25
BATCH2  = 64

history2 = model2.fit(
    x_train, y_train_oh,
    validation_data=(x_val, y_val_oh),
    epochs=EPOCHS2, batch_size=BATCH2,
    callbacks=callbacks2, verbose=1
)
```

## 14. Plot Curves

```
def plot_history(h, title_prefix="Model 2"):
    plt.figure(figsize=(10,4))
    # accuracy
    plt.subplot(1,2,1)
    plt.plot(h.history['accuracy'], label='train')
    plt.plot(h.history['val_accuracy'], label='val')
    plt.title(f"{title_prefix} Accuracy"); plt.xlabel("Epoch"); plt.ylabel("Acc"); plt.legend()
    # loss
    plt.subplot(1,2,2)
    plt.plot(h.history['loss'], label='train')
    plt.plot(h.history['val_loss'], label='val')
    plt.title(f"{title_prefix} Loss"); plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()
    plt.tight_layout(); plt.show()

plot_history(history2, "Model 2 (Enhanced)")
```

## 15. Evaluate + Confusion Matrix / Report

```
tr2_loss, tr2_acc = model2.evaluate(x_train, y_train_oh, verbose=0)
va2_loss, va2_acc = model2.evaluate(x_val,   y_val_oh,   verbose=0)
te2_loss, te2_acc = model2.evaluate(x_test,  y_test_oh,  verbose=0)

print(f"Model 2 — Accuracy  | Train: {tr2_acc:.4f}  Val: {va2_acc:.4f}  Test: {te2_acc:.4f}")
print(f"Model 2 — Loss      | Train: {tr2_loss:.4f}  Val: {va2_loss:.4f}  Test: {te2_loss:.4f}")

# Predictions for confusion matrix
y2_pred_probs = model2.predict(x_test, batch_size=256, verbose=0)
y2_pred = y2_pred_probs.argmax(axis=1)

cm2 = confusion_matrix(y_test, y2_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm2, annot=False, cmap='Greens', fmt='d',
            xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Model 2 — Confusion Matrix (Test)")
plt.tight_layout(); plt.show()

print("Model 2 — Classification Report (Test):")
print(classification_report(y_test, y2_pred, target_names=CLASS_NAMES))
```

## 16. Side-by-Side Comparison with Model 1

```
try:
    # requires tr_acc, va_acc, te_acc from Model 1 block to exist
    import pandas as pd
    cmp = pd.DataFrame({
        "Metric": ["Train Acc","Val Acc","Test Acc","Train Loss","Val Loss","Test Loss"],
        "Model 1 (Baseline)":[tr_acc, va_acc, te_acc, tr_loss, va_loss, te_loss],
        "Model 2 (Enhanced)":[tr2_acc, va2_acc, te2_acc, tr2_loss, va2_loss, te2_loss]
    })
    display(cmp)
except NameError:
    pass
```