```
# ====== Imports ======
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers, initializers
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.utils import to_categorical
```

```
# For reproducibility
tf.random.set_seed(42)
np.random.seed(42)
```

```
# ====== 1) Load CIFAR-10 ======
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
print("Train:", x_train.shape, y_train.shape, "Test:", x_test.shape, y_test.shape)
```

```
Train: (50000, 32, 32, 3) (50000, 1) Test: (10000, 32, 32, 3) (10000, 1)
```

```
# ====== 2) Preprocessing ======
# Normalize to [0,1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
```

```
# One-hot encode labels (for categorical_crossentropy)
num_classes = 10
y_train_cat = to_categorical(y_train, num_classes)
y_test_cat  = to_categorical(y_test, num_classes)
```

```
# Optionally create a validation split from training
val_split = 0.1
val_size = int(len(x_train) * val_split)
x_val = x_train[-val_size:]
y_val_cat = y_train_cat[-val_size:]
x_train2 = x_train[:-val_size]
y_train_cat2 = y_train_cat[:-val_size]

print("After split -> train:", x_train2.shape, "val:", x_val.shape)
```

```
After split -> train: (45000, 32, 32, 3) val: (5000, 32, 32, 3)
```

```
# ====== Data augmentation ======
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

datagen.fit(x_train2)
```

```
# ====== 3) Build the CNN model ======
def build_model(input_shape=(32,32,3), num_classes=10, activation='relu', weight_init='he_normal'):
    inputs = keras.Input(shape=input_shape)
    x = inputs

    # Block 1
    x = layers.Conv2D(32, (5,5), padding='same', kernel_initializer=weight_init)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation(activation)(x)
    x = layers.Conv2D(32, (3,3), padding='same', kernel_initializer=weight_init)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation(activation)(x)
    x = layers.MaxPooling2D((2,2))(x)
```

```
        x = layers.Dropout(0.25)(x)

        # Block 2
        x = layers.Conv2D(64, (5,5), padding='same', kernel_initializer=weight_init)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation(activation)(x)
        x = layers.Conv2D(64, (3,3), padding='same', kernel_initializer=weight_init)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation(activation)(x)
        x = layers.MaxPooling2D((2,2))(x)
        x = layers.Dropout(0.35)(x)

        # Block 3
        x = layers.Conv2D(128, (5,5), padding='same', kernel_initializer=weight_init)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation(activation)(x)
        x = layers.Conv2D(128, (3,3), padding='same', kernel_initializer=weight_init)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation(activation)(x)
        x = layers.MaxPooling2D((2,2))(x)
        x = layers.Dropout(0.4)(x)

        x = layers.Flatten()(x)
        x = layers.Dense(256, kernel_initializer=weight_init)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation(activation)(x)
        x = layers.Dropout(0.5)(x)

        outputs = layers.Dense(num_classes, activation='softmax')(x)
        model = models.Model(inputs, outputs)
        return model

    # Build with ReLU and He init (recommended)
    model = build_model(activation='relu', weight_init='he_normal')
    model.summary()
```

```
Model: "functional_3"
```

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 32, 32, 3)	0
conv2d_18 (Conv2D)	(None, 32, 32, 32)	2,432
batch_normalization_21 (BatchNormalization)	(None, 32, 32, 32)	128
activation_21 (Activation)	(None, 32, 32, 32)	0
conv2d_19 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_22 (BatchNormalization)	(None, 32, 32, 32)	128
activation_22 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_12 (Dropout)	(None, 16, 16, 32)	0
conv2d_20 (Conv2D)	(None, 16, 16, 64)	51,264
max_pooling2d_10 (MaxPooling2D)	(None, 8, 8, 64)	0
batch_normalization_27 (BatchNormalization)	(None, 256)	1,024
activation_27 (Activation)	(None, 256)	0
dropout_15 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 10)	2,570

```
# ====== 4) Compile: optimizer, loss, metrics ======
initial_lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=initial_lr)

model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# ====== 5) Callbacks ======
callbacks = [
    tf.keras.callbacks.ModelCheckpoint("best_cifar10_model.h5", save_best_only=True, monitor='val_accuracy', mode='max'),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, verbose=1),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True, verbose=1)
]
```

```
# ====== 6) Train the model======
batch_size = 64
epochs = 10

history = model.fit(
    datagen.flow(x_train2, y_train_cat2, batch_size=batch_size),
    steps_per_epoch=len(x_train2)//batch_size,
    epochs=epochs,
    validation_data=(x_val, y_val_cat),
    callbacks=callbacks
)
```

```
Epoch 1/10
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
84/703 ━━━━━━━━━━━━━━━━━━━━ 8:33 829ms/step - accuracy: 0.1845 - loss: 2.5251
```

```
# ====== 7) Plot training curves ======
def plot_history(h):
    plt.figure(figsize=(14,5))
    # accuracy
    plt.subplot(1,2,1)
    plt.plot(h.history['accuracy'], label='train_acc')
    plt.plot(h.history['val_accuracy'], label='val_acc')
    plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.legend(); plt.grid(True)
    # loss
    plt.subplot(1,2,2)
    plt.plot(h.history['loss'], label='train_loss')
    plt.plot(h.history['val_loss'], label='val_loss')
    plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend(); plt.grid(True)
    plt.show()

plot_history(history)
```

```
# ====== 8) Evaluate on test set ======
test_loss, test_acc = model.evaluate(x_test, y_test_cat, verbose=2)
print(f"Test accuracy: {test_acc:.4f}, Test loss: {test_loss:.4f}")

# Predictions & classification report
y_pred_probs = model.predict(x_test)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = y_test.flatten()
```

```
# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted'); plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
# Classification report (precision, recall, f1)
report = classification_report(y_true, y_pred, target_names=[
    'airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck'])
print(report)
```

```
# ====== 9) Save model and provide instructions for Colab/GitHub ======
model.save('cifar10_cnn_final.h5')
print("Model saved to cifar10_cnn_final.h5")
```

```
from google.colab import files
uploaded = files.upload()
```

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

# Replace 'your_image.jpg' with the uploaded file name
img_path = list(uploaded.keys())[0]

# Load & resize to 32x32
img = image.load_img(img_path, target_size=(32,32))

# Convert to array and normalize
img_array = image.img_to_array(img) / 255.0

# Expand dimensions to match model input (1,32,32,3)
img_array = np.expand_dims(img_array, axis=0)

# Predict
pred_probs = model.predict(img_array)
pred_class = np.argmax(pred_probs, axis=1)[0]
```

```
# Map class index to label
class_names = ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
print(f"Predicted class: {class_names[pred_class]}")

# Show the image
plt.imshow(img)
plt.axis('off')
plt.show()
```

Start coding or generate with AI.

Start coding or generate with AI.