

```

import tensorflow as tf
from tensorflow import keras
import os

# Option 1: Download CIFAR-10 dataset (built-in)
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Option 2: Download custom dataset from Kaggle/URL
# !kaggle datasets download -d dataset-name

# Option 3: Use TensorFlow Datasets
# import tensorflow_datasets as tfds
# dataset = tfds.load('cats_vs_dogs', split=['train', 'test'], as_supervised=True)

print(f"Training samples: {len(x_train)}")
print(f"Testing samples: {len(x_test)}")

from sklearn.model_selection import train_test_split
import numpy as np

# Split training data into train and validation
x_train, x_val, y_train, y_val = train_test_split(
    x_train, y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train # Maintain class distribution
)

print(f"Training set: {len(x_train)}")
print(f"Validation set: {len(x_val)}")
print(f"Test set: {len(x_test)}")

# Normalize pixel values
x_train = x_train.astype('float32') / 255.0
x_val = x_val.astype('float32') / 255.0

```

```
x_test = x_test.astype('float32') / 255.0
```

```
from tensorflow.keras import layers
```

```
# Create data augmentation layer
```

```
data_augmentation = keras.Sequential([  
    layers.RandomFlip("horizontal"),  
    layers.RandomRotation(0.1),  
    layers.RandomZoom(0.1),  
    layers.RandomContrast(0.1),  
    layers.RandomTranslation(0.1, 0.1),  
], name="data_augmentation")
```

```
# Alternative: Using ImageDataGenerator
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
datagen = ImageDataGenerator(  
    rotation_range=20,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    horizontal_flip=True,  
    zoom_range=0.2,  
    shear_range=0.2,  
    fill_mode='nearest'  
)
```

```
# Visualize augmented images
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(10, 10))
```

```
for i in range(9):
```

```
    augmented_image = data_augmentation(x_train[i:i+1])
```

```
    plt.subplot(3, 3, i + 1)
```

```
    plt.imshow(augmented_image[0])
```

```
    plt.axis('off')
```

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

```
# Build the model
```

```
def create_model(input_shape, num_classes):
```

```
    model = keras.Sequential([
        data_augmentation,

        layers.Conv2D(32, 3, activation='relu', padding='same', input_shape=input_shape),
        layers.BatchNormalization(),
        layers.Conv2D(32, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(),
        layers.Dropout(0.25),

        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(),
        layers.Dropout(0.25),

        layers.Conv2D(128, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(128, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(),
        layers.Dropout(0.25),

        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
```

```

    return model

# Create and compile model
input_shape = x_train.shape[1:]
num_classes = 10 # For CIFAR-10

model = create_model(input_shape, num_classes)

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

# Callbacks
callbacks = [
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-7
    ),
    keras.callbacks.ModelCheckpoint(
        'best_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        mode='max'
    )
]

```

```

# Train the model
history = model.fit(
    x_train, y_train,
    batch_size=64,
    epochs=50,
    validation_data=(x_val, y_val),
    callbacks=callbacks
)

# Evaluate on test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")

# Make predictions
predictions = model.predict(x_test)
predicted_classes = np.argmax(predictions, axis=1)

# Plot training history
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

```

```
plt.title('Model Loss')
```

```
plt.tight_layout()
```

```
plt.show()
```

```
# Classification Report
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
import seaborn as sns
```

```
print(classification_report(y_test, predicted_classes))
```

```
# Confusion Matrix
```

```
cm = confusion_matrix(y_test, predicted_classes)
```

```
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()
```

```
# For custom image folders
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Directory structure:
```

```
# data/
```

```
#   train/
```

```
#     class1/
```

```
#     class2/
```

```
#   validation/
```

```
#     class1/
```

```
#     class2/
```

```
#   test/
```

```
#     class1/
```

```
#     class2/
```

```

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    'data/validation',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    'data/test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

# Train with generator
history = model.fit(

```

```
train_generator,  
epochs=50,  
validation_data=val_generator,  
callbacks=callbacks  
)
```

```
pip install tensorflow scikit-learn matplotlib seaborn numpy pandas
```