**AlexNet**

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Normalize the data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# One-hot encode the labels

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Define the AlexNet model

def alexnet():

model = models.Sequential()

# Layer 1: Convolution + ReLU + MaxPooling

model.add(layers.Conv2D(96, (11, 11), strides=1, activation='relu', input\_shape=(32, 32, 3), padding='same'))

model.add(layers.MaxPooling2D(pool\_size=(3, 3), strides=2))

# Layer 2: Convolution + ReLU + MaxPooling

model.add(layers.Conv2D(256, (5, 5), activation='relu', padding='same'))

model.add(layers.MaxPooling2D(pool\_size=(3, 3), strides=2))

# Layer 3: Convolution + ReLU

model.add(layers.Conv2D(384, (3, 3), activation='relu', padding='same'))

# Layer 4: Convolution + ReLU

model.add(layers.Conv2D(384, (3, 3), activation='relu', padding='same'))

# Layer 5: Convolution + ReLU + MaxPooling

model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))

model.add(layers.MaxPooling2D(pool\_size=(3, 3), strides=2))

# Flatten the output

model.add(layers.Flatten())

# Fully Connected Layers

model.add(layers.Dense(4096, activation='relu'))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(4096, activation='relu'))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax')) # Output layer

model.summary()

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

# Instantiate the model

model = alexnet()

# Train the model

history = model.fit(x\_train, y\_train, epochs=1, validation\_split=0.2, batch\_size=128, verbose=1)

# Plot Training and Validation Accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Evaluate the model on test data

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy\*100:.2f}%")

**import tensorflow as tf**

**from tensorflow.keras import layers, models**

**from tensorflow.keras.applications import VGG16**

**from tensorflow.keras.datasets import cifar10**

**from tensorflow.keras.utils import to\_categorical**

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

**import matplotlib.pyplot as plt**

**# Load CIFAR-10 dataset**

**(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()**

**# Normalize the data**

**x\_train = x\_train.astype('float32') / 255.0**

**x\_test = x\_test.astype('float32') / 255.0**

**# One-hot encode the labels**

**y\_train = to\_categorical(y\_train, 10)**

**y\_test = to\_categorical(y\_test, 10)**

**# Create VGG16-based model**

**def create\_vgg\_base\_model():**

**base\_model = VGG16(weights=None, include\_top=False, input\_shape=(32, 32, 3))**

**model = models.Sequential()**

**model.add(base\_model)**

**model.add(layers.Flatten())**

**model.add(layers.Dense(512, activation='relu'))**

**model.add(layers.Dropout(0.5))**

**model.add(layers.Dense(10, activation='softmax'))**

**return model**

**# Define a LeNet model**

**def create\_lenet\_model():**

**model = models.Sequential()**

**model.add(layers.Conv2D(6, (5, 5), activation='relu', input\_shape=(32, 32, 3)))**

**model.add(layers.AveragePooling2D((2, 2)))**

**model.add(layers.Conv2D(16, (5, 5), activation='relu'))**

**model.add(layers.AveragePooling2D((2, 2)))**

**model.add(layers.Flatten())**

**model.add(layers.Dense(120, activation='relu'))**

**model.add(layers.Dense(84, activation='relu'))**

**model.add(layers.Dense(10, activation='softmax'))**

**return model**

**# Define an AlexNet model**

**def create\_alexnet\_model():**

**model = models.Sequential()**

**model.add(layers.Conv2D(96, (11, 11), strides=4, activation='relu', input\_shape=(32, 32, 3)))**

**model.add(layers.MaxPooling2D((3, 3), strides=2))**

**model.add(layers.Conv2D(256, (5, 5), activation='relu', padding='same'))**

**model.add(layers.MaxPooling2D((3, 3), strides=2))**

**model.add(layers.Conv2D(384, (3, 3), activation='relu', padding='same'))**

**model.add(layers.Conv2D(384, (3, 3), activation='relu', padding='same'))**

**model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))**

**model.add(layers.MaxPooling2D((3, 3), strides=2))**

**model.add(layers.Flatten())**

**model.add(layers.Dense(4096, activation='relu'))**

**model.add(layers.Dropout(0.5))**

**model.add(layers.Dense(4096, activation='relu'))**

**model.add(layers.Dropout(0.5))**

**model.add(layers.Dense(10, activation='softmax'))**

**return model**

**# Choose a model to train**

**model\_choice = "vgg" # Change to "lenet" or "alexnet" as needed**

**if model\_choice == "vgg":**

**model = create\_vgg\_base\_model()**

**elif model\_choice == "lenet":**

**model = create\_lenet\_model()**

**elif model\_choice == "alexnet":**

**model = create\_alexnet\_model()**

**else:**

**raise ValueError("Invalid model choice. Choose 'vgg', 'lenet', or 'alexnet'.")**

**# Compile the model**

**model.compile(optimizer='adam',**

**loss='categorical\_crossentropy',**

**metrics=['accuracy'])**

**# Data Augmentation**

**data\_gen = ImageDataGenerator(**

**rotation\_range=15,**

**width\_shift\_range=0.1,**

**height\_shift\_range=0.1,**

**horizontal\_flip=True**

**)**

**data\_gen.fit(x\_train)**

**# Train the model**

**history = model.fit(data\_gen.flow(x\_train, y\_train, batch\_size=64),**

**epochs=50,**

**validation\_data=(x\_test, y\_test),**

**verbose=1)**

**# Evaluate the model**

**test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=2)**

**print(f"Test accuracy: {test\_accuracy}")**

**# Plot training and validation metrics**

**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='Validation Accuracy')**

**plt.title('Accuracy')**

**plt.legend()**

**plt.subplot(1, 2, 2)**

**plt.plot(history.history['loss'], label='Train Loss')**

**plt.plot(history.history['val\_loss'], label='Validation Loss')**

**plt.title('Loss')**

**plt.legend()**

**plt.show()**

**# Check for overfitting or underfitting**

**train\_accuracy = history.history['accuracy'][-1]**

**val\_accuracy = history.history['val\_accuracy'][-1]**

**diff = train\_accuracy - val\_accuracy**

**if diff > 0.05:**

**print("The model is overfitting. Consider adding more regularization techniques or reducing model complexity.")**

**elif diff < -0.05:**

**print("The model is underfitting. Consider increasing model complexity or training longer.")**

**else:**

**print("The model is a good fit.")**