**Assignment 3**

**Build the Image classification model by dividing the model.**

**Step 1: Loading and Preprocessing the Image Data**

We’ll use TensorFlow/Keras and the ImageDataGenerator class to load and preprocess the image dataset.

*import tensorflow as tf*

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

*from sklearn.model\_selection import train\_test\_split*

*# Directory path to your image dataset (organized into subfolders for each class)*

*DATASET\_PATH = 'path/to/your/dataset'*

*# Split parameters*

*IMAGE\_SIZE = (128, 128) # Resize all images to 128x128*

*BATCH\_SIZE = 32 # Number of images processed together*

*SEED = 42 # Ensure reproducibility*

*# Create ImageDataGenerator objects for data augmentation*

*datagen = ImageDataGenerator(*

*rescale=1.0/255.0, # Normalize pixel values between 0 and 1*

*validation\_split=0.2, # Use 20% of data for validation*

*rotation\_range=15, # Randomly rotate images*

*width\_shift\_range=0.1, # Shift width randomly*

*height\_shift\_range=0.1, # Shift height randomly*

*horizontal\_flip=True # Random horizontal flip*

*)*

*# Load and split data into training and validation sets*

*train\_data = datagen.flow\_from\_directory(*

*DATASET\_PATH,*

*target\_size=IMAGE\_SIZE,*

*batch\_size=BATCH\_SIZE,*

*class\_mode='categorical', # Use 'categorical' for multi-class classification*

*subset='training',*

*seed=SEED*

*)*

*val\_data = datagen.flow\_from\_directory(*

*DATASET\_PATH,*

*target\_size=IMAGE\_SIZE,*

*batch\_size=BATCH\_SIZE,*

*class\_mode='categorical',*

*subset='validation',*

*seed=SEED*

*)*

**ImageDataGenerator:** Helps in loading and augmenting the dataset.

**Data Augmentation:** Adds randomness (rotation, shifts, flips) to prevent overfitting.

**Normalization:** We rescale pixel values to be between 0 and 1.

**Train-Validation Split:** 80% data for training, 20% for validation to check model generalization.

**Step 2: Defining the Model’s Architecture**

We’ll build a **Convolutional Neural Network (CNN)** for image classification.

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout*

*# Build the CNN model*

*model = Sequential([*

*Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)), # 1st Convolutional layer*

*MaxPooling2D(pool\_size=(2, 2)), # Max Pooling*

*Conv2D(64, (3, 3), activation='relu'), # 2nd Convolutional layer*

*MaxPooling2D(pool\_size=(2, 2)),*

*Conv2D(128, (3, 3), activation='relu'), # 3rd Convolutional layer*

*MaxPooling2D(pool\_size=(2, 2)),*

*Flatten(), # Flatten feature maps to 1D vector*

*Dense(128, activation='relu'), # Fully connected layer*

*Dropout(0.5), # Dropout for regularization*

*Dense(train\_data.num\_classes, activation='softmax') # Output layer with softmax*

*])*

*# Compile the model*

*model.compile(*

*optimizer='adam', # Adam optimizer for faster convergence*

*loss='categorical\_crossentropy', # Suitable loss for multi-class classification*

*metrics=['accuracy']*

*)*

*# Summary of the model*

*model.summary()*

**Convolutional Layers:** Extract features from input images.

**MaxPooling:** Reduces spatial dimensions to prevent overfitting.

**Flatten Layer:** Converts feature maps to a 1D vector for Dense layers.

**Dropout:** Prevents overfitting by randomly disabling some neurons.

**Softmax Layer:** Gives probabilities for each class.

**Adam Optimizer:** Adaptive optimizer for efficient training.

**Step 3: Training the Model**

We’ll train the model using the training and validation data generators.

*# Train the model*

*history = model.fit(*

*train\_data,*

*validation\_data=val\_data,*

*epochs=10, # Number of epochs*

*verbose=1 # Display training progress*

*)*

**Epochs:** Controls how many times the model sees the entire dataset.

**Training Data:** The model learns from this dataset.

**Validation Data:** Used to evaluate the model after each epoch.

**Step 4: Estimating the Model’s Performance**

After training, we’ll evaluate the model on the validation data.

*import matplotlib.pyplot as plt*

*# Evaluate the model on validation data*

*val\_loss, val\_accuracy = model.evaluate(val\_data)*

*print(f"Validation Loss: {val\_loss:.4f}")*

*print(f"Validation Accuracy: {val\_accuracy:.4f}")*

*# Plot training history (accuracy and loss)*

*def plot\_history(history):*

*plt.figure(figsize=(12, 4))*

*# Plot accuracy*

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

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

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

*plt.title('Model Accuracy')*

*plt.xlabel('Epochs')*

*plt.ylabel('Accuracy')*

*plt.legend()*

*# Plot loss*

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

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

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

*plt.title('Model Loss')*

*plt.xlabel('Epochs')*

*plt.ylabel('Loss')*

*plt.legend()*

*plt.show()*

*# Call the function to plot training history*

*plot\_history(history)*

**Model Evaluation:** We compute loss and accuracy on the validation dataset.

**Training History Plot:** Visualizes how the model’s accuracy and loss change over epochs.