**Artificial Neural networks**

**Classification Using Artificial Neural Networks with Hyperparameter Tuning on Alphabets Data**

**Overview**

**In this assignment, you will be tasked with developing a classification model using Artificial Neural Networks (ANNs) to classify data points from the "Alphabets\_data.csv" dataset into predefined categories of alphabets. This exercise aims to deepen your understanding of ANNs and the significant role hyperparameter tuning plays in enhancing model performance.**

**Dataset: "Alphabets\_data.csv"**

**The dataset provided, "Alphabets\_data.csv", consists of labeled data suitable for a classification task aimed at identifying different alphabets. Before using this data in your model, you'll need to preprocess it to ensure optimal performance.**

**Tasks**

**1. Data Exploration and Preprocessing**

* **Begin by loading and exploring the "Alphabets\_data.csv" dataset. Summarize its key features such as the number of samples, features, and classes.**
* **Execute necessary data preprocessing steps including data normalization, managing missing values.**

**Answer :**

**Code used :**

import pandas as pd

import pprint

import os

# --- File paths ---

input\_path = r"D:\DATA SCIENCE\ASSIGNMENTS\18 neural networks\Neural networks\Alphabets\_data.csv"

output\_folder = r"d:\DATA SCIENCE\ASSIGNMENTS\18 neural networks\Neural networks"

output\_file = os.path.join(output\_folder, "dataset\_overview.txt")

# --- Load dataset ---

alphabets\_data\_df = pd.read\_csv(input\_path)

# --- Dataset summary ---

dataset\_overview = {

    "Shape": alphabets\_data\_df.shape,

    "Column\_Names": alphabets\_data\_df.columns.tolist(),

    "Missing\_Values": alphabets\_data\_df.isnull().sum().to\_dict(),

    "Data\_Types": alphabets\_data\_df.dtypes.astype(str).to\_dict(),

    "Sample\_Rows": alphabets\_data\_df.head().to\_dict(orient="records")

}

# --- Save output to file ---

with open(output\_file, "w") as f:

    f.write("=== DATASET OVERVIEW ===\n\n")

    pprint.pprint(dataset\_overview, stream=f, sort\_dicts=False)

print(f"✅ Dataset summary saved successfully at:\n{output\_file}")

**=== DATASET OVERVIEW ===**

**{'Shape': (20000, 17),**

**'Column\_Names': ['letter',**

**'xbox',**

**'ybox',**

**'width',**

**'height',**

**'onpix',**

**'xbar',**

**'ybar',**

**'x2bar',**

**'y2bar',**

**'xybar',**

**'x2ybar',**

**'xy2bar',**

**'xedge',**

**'xedgey',**

**'yedge',**

**'yedgex'],**

**'Missing\_Values': {'letter': 0,**

**'xbox': 0,**

**'ybox': 0,**

**'width': 0,**

**'height': 0,**

**'onpix': 0,**

**'xbar': 0,**

**'ybar': 0,**

**'x2bar': 0,**

**'y2bar': 0,**

**'xybar': 0,**

**'x2ybar': 0,**

**'xy2bar': 0,**

**'xedge': 0,**

**'xedgey': 0,**

**'yedge': 0,**

**'yedgex': 0},**

**'Data\_Types': {'letter': 'object',**

**'xbox': 'int64',**

**'ybox': 'int64',**

**'width': 'int64',**

**'height': 'int64',**

**'onpix': 'int64',**

**'xbar': 'int64',**

**'ybar': 'int64',**

**'x2bar': 'int64',**

**'y2bar': 'int64',**

**'xybar': 'int64',**

**'x2ybar': 'int64',**

**'xy2bar': 'int64',**

**'xedge': 'int64',**

**'xedgey': 'int64',**

**'yedge': 'int64',**

**'yedgex': 'int64'},**

**'Sample\_Rows': [{'letter': 'T',**

**'xbox': 2,**

**'ybox': 8,**

**'width': 3,**

**'height': 5,**

**'onpix': 1,**

**'xbar': 8,**

**'ybar': 13,**

**'x2bar': 0,**

**'y2bar': 6,**

**'xybar': 6,**

**'x2ybar': 10,**

**'xy2bar': 8,**

**'xedge': 0,**

**'xedgey': 8,**

**'yedge': 0,**

**'yedgex': 8},**

**{'letter': 'I',**

**'xbox': 5,**

**'ybox': 12,**

**'width': 3,**

**'height': 7,**

**'onpix': 2,**

**'xbar': 10,**

**'ybar': 5,**

**'x2bar': 5,**

**'y2bar': 4,**

**'xybar': 13,**

**'x2ybar': 3,**

**'xy2bar': 9,**

**'xedge': 2,**

**'xedgey': 8,**

**'yedge': 4,**

**'yedgex': 10},**

**{'letter': 'D',**

**'xbox': 4,**

**'ybox': 11,**

**'width': 6,**

**'height': 8,**

**'onpix': 6,**

**'xbar': 10,**

**'ybar': 6,**

**'x2bar': 2,**

**'y2bar': 6,**

**'xybar': 10,**

**'x2ybar': 3,**

**'xy2bar': 7,**

**'xedge': 3,**

**'xedgey': 7,**

**'yedge': 3,**

**'yedgex': 9},**

**{'letter': 'N',**

**'xbox': 7,**

**'ybox': 11,**

**'width': 6,**

**'height': 6,**

**'onpix': 3,**

**'xbar': 5,**

**'ybar': 9,**

**'x2bar': 4,**

**'y2bar': 6,**

**'xybar': 4,**

**'x2ybar': 4,**

**'xy2bar': 10,**

**'xedge': 6,**

**'xedgey': 10,**

**'yedge': 2,**

**'yedgex': 8},**

**{'letter': 'G',**

**'xbox': 2,**

**'ybox': 1,**

**'width': 3,**

**'height': 1,**

**'onpix': 1,**

**'xbar': 8,**

**'ybar': 6,**

**'x2bar': 6,**

**'y2bar': 6,**

**'xybar': 6,**

**'x2ybar': 5,**

**'xy2bar': 9,**

**'xedge': 1,**

**'xedgey': 7,**

**'yedge': 5,**

**'yedgex': 10}]}**

**2. Model Implementation**

* **Construct a basic ANN model using your chosen high-level neural network library. Ensure your model includes at least one hidden layer.**
* **Divide the dataset into training and test sets.**
* **Train your model on the training set and then use it to make predictions on the test set.**

**Answer :**

Python script that:

* Loads Alphabets\_data.csv (your path or the uploaded /mnt/data copy),
* Preprocesses (label encoding → one-hot, StandardScaler),
* Splits data (train/test),
* Builds a simple Keras ANN with **one hidden layer** (plus optional deeper variant),
* Trains the model with EarlyStopping,
* Saves trained model, training history (.png) and test predictions (.csv) into your folder d:/python apps/neural networks,
* Prints evaluation metrics (accuracy, precision, recall, F1) and confusion matrix.

Drop this into a .py file (e.g., ann\_alphabets.py) and run it in your venv. If you prefer I can tweak architecture/hyperparams after you peek at results.

**Code used :**

**"""**

**ANN for Alphabets classification (basic implementation)**

**- Expects Alphabets\_data.csv available at input\_path (change if needed)**

**- Saves outputs (model, metrics, plots, preds) to output\_folder**

**- Uses TensorFlow / Keras**

**Requirements:**

**pip install numpy pandas scikit-learn matplotlib tensorflow**

**"""**

**import os**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import LabelEncoder, StandardScaler**

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

**from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, Dropout**

**from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint**

**# -----------------------**

**# Paths - change if needed**

**# -----------------------**

**# If you want to use the copy uploaded to this environment, set input\_path = "/mnt/data/Alphabets\_data.csv"**

**input\_path = r"D:\DATA SCIENCE\ASSIGNMENTS\18 neural networks\Neural networks\Alphabets\_data.csv"**

**# Save outputs to this folder (as you requested)**

**output\_folder = r"d:\python apps\neural networks"**

**os.makedirs(output\_folder, exist\_ok=True)**

**# -----------------------**

**# Load dataset**

**# -----------------------**

**df = pd.read\_csv(input\_path)**

**print(f"Loaded data: {df.shape[0]} rows, {df.shape[1]} columns")**

**# -----------------------**

**# Preprocessing**

**# -----------------------**

**# 1) Separate X and y**

**target\_col = "letter"**

**if target\_col not in df.columns:**

**raise ValueError(f"Target column '{target\_col}' not found in CSV.")**

**X = df.drop(columns=[target\_col])**

**y = df[target\_col]**

**# 2) Label encode target and one-hot encode for Keras**

**le = LabelEncoder()**

**y\_int = le.fit\_transform(y) # integers 0..25**

**num\_classes = len(le.classes\_)**

**y\_cat = tf.keras.utils.to\_categorical(y\_int, num\_classes=num\_classes)**

**print(f"Detected {num\_classes} classes: {list(le.classes\_)}")**

**# 3) Feature scaling**

**scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

**# 4) Train-test split (stratify to keep class balance)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(**

**X\_scaled, y\_cat, test\_size=0.2, random\_state=42, stratify=y\_int**

**)**

**print("Train shape:", X\_train.shape, "Test shape:", X\_test.shape)**

**# Optionally save the scaler and label encoder for later inference**

**import joblib**

**joblib.dump(scaler, os.path.join(output\_folder, "scaler.joblib"))**

**joblib.dump(le, os.path.join(output\_folder, "label\_encoder.joblib"))**

**# -----------------------**

**# Build model (basic: one hidden layer)**

**# -----------------------**

**input\_dim = X\_train.shape[1]**

**hidden\_units = 128 # sensible default for 16 features**

**dropout\_rate = 0.2**

**def build\_basic\_ann(input\_dim, hidden\_units=128, dropout\_rate=0.2, num\_classes=26):**

**model = Sequential([**

**Dense(hidden\_units, input\_dim=input\_dim, activation="relu"),**

**Dropout(dropout\_rate),**

**Dense(num\_classes, activation="softmax")**

**])**

**return model**

**model = build\_basic\_ann(input\_dim, hidden\_units, dropout\_rate, num\_classes)**

**model.summary()**

**# -----------------------**

**# Compile & callbacks**

**# -----------------------**

**learning\_rate = 0.001**

**model.compile(**

**optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),**

**loss="categorical\_crossentropy",**

**metrics=["accuracy"]**

**)**

**checkpoint\_path = os.path.join(output\_folder, "best\_ann\_model.h5")**

**callbacks = [**

**EarlyStopping(monitor="val\_loss", patience=8, restore\_best\_weights=True, verbose=1),**

**ModelCheckpoint(filepath=checkpoint\_path, monitor="val\_loss", save\_best\_only=True, verbose=1)**

**]**

**# -----------------------**

**# Train**

**# -----------------------**

**batch\_size = 64**

**epochs = 80**

**history = model.fit(**

**X\_train, y\_train,**

**validation\_split=0.15,**

**epochs=epochs,**

**batch\_size=batch\_size,**

**callbacks=callbacks,**

**verbose=2**

**)**

**# Save final model (already best saved by checkpoint)**

**final\_model\_path = os.path.join(output\_folder, "final\_ann\_model.h5")**

**model.save(final\_model\_path)**

**print(f"Saved model to: {final\_model\_path} (best checkpoint at {checkpoint\_path})")**

**# -----------------------**

**# Plot training history**

**# -----------------------**

**plt.figure(figsize=(10,4))**

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

**plt.plot(history.history["loss"], label="train\_loss")**

**plt.plot(history.history["val\_loss"], label="val\_loss")**

**plt.xlabel("epoch"); plt.title("Loss"); plt.legend()**

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

**plt.plot(history.history["accuracy"], label="train\_acc")**

**plt.plot(history.history["val\_accuracy"], label="val\_acc")**

**plt.xlabel("epoch"); plt.title("Accuracy"); plt.legend()**

**plt.tight\_layout()**

**plt.savefig(os.path.join(output\_folder, "training\_history.png"))**

**plt.close()**

**print("Saved training history plot to training\_history.png")**

**# -----------------------**

**# Evaluate on test set**

**# -----------------------**

**y\_pred\_prob = model.predict(X\_test)**

**y\_pred\_int = np.argmax(y\_pred\_prob, axis=1)**

**y\_true\_int = np.argmax(y\_test, axis=1)**

**acc = accuracy\_score(y\_true\_int, y\_pred\_int)**

**print(f"\nTest accuracy: {acc:.4f}")**

**# Classification report (per-class precision/recall/f1)**

**report = classification\_report(y\_true\_int, y\_pred\_int, target\_names=le.classes\_, digits=4)**

**print("\nClassification Report:\n", report)**

**# Save classification report to file**

**with open(os.path.join(output\_folder, "classification\_report.txt"), "w") as f:**

**f.write(f"Test accuracy: {acc:.4f}\n\n")**

**f.write(report)**

**# Confusion matrix (save as CSV and plot)**

**cm = confusion\_matrix(y\_true\_int, y\_pred\_int)**

**cm\_df = pd.DataFrame(cm, index=le.classes\_, columns=le.classes\_)**

**cm\_df.to\_csv(os.path.join(output\_folder, "confusion\_matrix.csv"))**

**plt.figure(figsize=(12,10))**

**plt.imshow(cm, interpolation="nearest")**

**plt.title("Confusion matrix")**

**plt.colorbar()**

**plt.xlabel("Predicted")**

**plt.ylabel("True")**

**plt.xticks(range(len(le.classes\_)), le.classes\_, rotation=90)**

**plt.yticks(range(len(le.classes\_)), le.classes\_)**

**plt.tight\_layout()**

**plt.savefig(os.path.join(output\_folder, "confusion\_matrix.png"))**

**plt.close()**

**print("Saved confusion matrix files")**

**# -----------------------**

**# Save predictions (human readable)**

**# -----------------------**

**pred\_df = pd.DataFrame({**

**"true\_label": le.inverse\_transform(y\_true\_int),**

**"pred\_label": le.inverse\_transform(y\_pred\_int),**

**"pred\_confidence": np.max(y\_pred\_prob, axis=1)**

**})**

**pred\_df.to\_csv(os.path.join(output\_folder, "test\_predictions.csv"), index=False)**

**print("Saved test predictions to test\_predictions.csv")**

**print("\n✅ All outputs saved in:", output\_folder)**

**Quick notes & tips**

* This is a **basic** architecture (one hidden layer + dropout). If accuracy is low, try:
  + Increasing hidden\_units (256, 512), adding more hidden layers (2–3 layers), or changing activation (ReLU is fine).
  + Adjusting learning\_rate, batch\_size, or optimizer (Adam works well).
  + Running class\_weight or focal loss if classes become imbalanced (but here you should be fine with stratified split).
* For **hyperparameter tuning** use Keras Tuner or sklearn RandomizedSearchCV wrapper (we’ll do that later in Task 3).
* If TensorFlow isn’t installed in your venv, install it with pip install tensorflow (or pip install tensorflow-cpu if you want the CPU-only variant).
* The script saves:
  + best\_ann\_model.h5 — best model checkpoint (based on val\_loss),
  + final\_ann\_model.h5 — final model after training,
  + training\_history.png — loss/accuracy curves,
  + classification\_report.txt and confusion\_matrix.csv/png,
  + test\_predictions.csv,
  + scaler.joblib and label\_encoder.joblib for later inference.

**3. Hyperparameter Tuning**

* **Modify various hyperparameters, such as the number of hidden layers, neurons per hidden layer, activation functions, and learning rate, to observe their impact on model performance.**
* **Adopt a structured approach like grid search or random search for hyperparameter tuning, documenting your methodology thoroughly.**

**4. Evaluation**

* **Employ suitable metrics such as accuracy, precision, recall, and F1-score to evaluate your model's performance.**
* **Discuss the performance differences between the model with default hyperparameters and the tuned model, emphasizing the effects of hyperparameter tuning.**

**Evaluation Criteria**

* **Accuracy and completeness of the implementation.**
* **Proficiency in data preprocessing and model development.**
* **Systematic approach and thoroughness in hyperparameter tuning.**
* **Depth of evaluation and discussion.**
* **Overall quality of the report.**

**Additional Resources**

* [**TensorFlow Documentation**](https://www.tensorflow.org/)
* [**Keras Documentation**](https://keras.io/)

**We wish you the best of luck with this assignment. Enjoy exploring the fascinating world of neural networks and the power of hyperparameter tuning!**