TensorFlow Setup & Project Organization Guide

Project Structure

```
Salary_Predictor/
├── salary_data_small.csv
├── salary_predictor_actual_tf.py
├── tf_env/                # Virtual environment
└── README.md
```

Step 1: Create Project Folder

```
# Create project folder on Desktop
mkdir "Salary_Predictor"
cd "Salary_Predictor"
```

Step 2: Set Up Virtual Environment

```
# Create virtual environment
py -3.13 -m venv tf_env

# Activate virtual environment
tf_env\Scripts\activate
```

Step 3: Install Required Packages

```
# Install all packages in one command
pip install tensorflow pandas numpy matplotlib scikit-learn joblib
jupyterlab

# Verify installation
python -c "import tensorflow as tf; print('TensorFlow version:',
tf.__version__)"
```

Step 4: Prepare Project Files

Files to Include:

1. **salary_data_small.csv** — Dataset file
2. **salary_predictor_actual_tf.py** — TensorFlow training and prediction script

---

📄 Sample Dataset (`salary_data_small.csv`)

```
years_experience,education_level,salary_usd
0,HighSchool,"32,000"
1,HighSchool,"36,000"
3,HighSchool,"42,000"
5,HighSchool,"48,500"
8,HighSchool,"56,000"
1,Bachelor,"45,000"
3,Bachelor,"52,000"
5,Bachelor,"60,000"
8,Bachelor,"72,000"
10,Bachelor,"80,000"
1,Master,"52,000"
3,Master,"60,000"
5,Master,"70,000"
8,Master,"85,000"
12,Master,"105,000"
2,PhD,"70,000"
4,PhD,"82,000"
6,PhD,"96,000"
9,PhD,"120,000"
15,PhD,"160,000"
```

💻 Script: `salary_predictor_actual_tf.py`

```
import tensorflow as tf

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import matplotlib

matplotlib.use('TkAgg')  # Use Tkinter backend

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')


print("🚀 Starting Salary Prediction Model...")


# Set random seeds for reproducibility
```

```
tf.random.set_seed(42)
np.random.seed(42)

print("📊 Loading and preparing your dataset...")

# Load your actual dataset
def load_salary_data(file_path):
    """Load and preprocess the salary dataset"""
    # Read the CSV file
    data = pd.read_csv(file_path)

    # Clean the salary column - remove commas and convert to float
    data['salary_usd'] = data['salary_usd'].astype(str).str.replace(',', '').astype(float)

    print(f"Dataset loaded: {data.shape[0]} samples")
    print(f"Columns: {list(data.columns)}")

    return data

# Load your data
try:
    data = load_salary_data('salary_data_small.csv')
    print("☑ CSV file loaded successfully!")
except FileNotFoundError:
    print("❌ ERROR: salary_data_small.csv file not found!")
    print("Please make sure salary_data_small.csv is in the same directory")
    exit()
```

```
print("\n📊 Dataset Overview:")
print(data.head(10))

print("\n🔧 Preprocessing data...")

# Encode education level with proper ordering
education_order = ['HighSchool', 'Bachelor', 'Master', 'PhD']
education_mapping = {level: i for i, level in enumerate(education_order)}
data['education_encoded'] = data['education_level'].map(education_mapping)

print("Education level mapping:")
for level, code in education_mapping.items():
    print(f"  {level}: {code}")

# Prepare features and target
X = data[['years_experience', 'education_encoded']].values
y = data['salary_usd'].values

print(f"\nFeature ranges:")
print(f"Years Experience: {X[:, 0].min()} - {X[:, 0].max()} years")
print(f"Salary Range: ${y.min():,.0f} - ${y.max():,.0f}")

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"\nTraining set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Build a simpler model suitable for small dataset
print("\n🤖 Building TensorFlow model...")

model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(2,)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)

print("Model architecture:")
```

```
model.summary()

# Train the model and capture history
print("\n🚀 Training model...")

history = model.fit(
    X_train_scaled, y_train,
    epochs=200,
    batch_size=4,
    validation_data=(X_test_scaled, y_test),
    verbose=1
)

# Evaluate the model
print("\n📊 Evaluating model...")
train_loss, train_mae = model.evaluate(X_train_scaled, y_train, verbose=0)
test_loss, test_mae = model.evaluate(X_test_scaled, y_test, verbose=0)

print(f"Training MAE: ${train_mae:,.2f}")
print(f"Test MAE: ${test_mae:,.2f}")

# Make predictions
print("\n🔮 Making predictions...")
predictions = model.predict(X_test_scaled, verbose=0).flatten()

print("\nTest set predictions vs actual:")
print("=" * 50)
```

```
for i, (actual, pred) in enumerate(zip(y_test, predictions)):
    experience = X_test[i][0]
    education_code = int(X_test[i][1])
    education_level = education_order[education_code]
    difference = abs(actual - pred)

    print(f" {experience:2.0f} yrs, {education_level:10} | "
          f"Actual: ${actual:>7,.0f} | Pred: ${pred:>7,.0f}")

# Create prediction function
def predict_salary(years_experience, education_level):
    """Predict salary for given experience and education level"""
    if education_level not in education_mapping:
        raise ValueError(f"Education level must be one of {list(education_mapping.keys())}")

    # Encode education level
    education_encoded = education_mapping[education_level]

    # Prepare input
    input_data = np.array([[years_experience, education_encoded]])
    input_scaled = scaler.transform(input_data)

    # Make prediction
    prediction = model.predict(input_scaled, verbose=0)[0][0]

    return prediction
```

```
# Test predictions
print("\n🎯 Salary Predictions:")
print("=" * 40)
test_cases = [
    (5, 'Bachelor'),
    (8, 'Master'),
    (3, 'PhD'),
    (2, 'HighSchool')
]

for years, education in test_cases:
    predicted = predict_salary(years, education)
    print(f" {years} years, {education:10}: ${predicted:,.0f}")

# ================================
# 🎨 NOW PLOT THE GRAPHS (AFTER TRAINING)
# ================================

print("\n📊 Creating visualizations...")

# Plot training history and save to file
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE')
plt.xlabel('Epoch')
plt.ylabel('MAE ($)')
plt.legend()

plt.tight_layout()
plt.savefig('training_history.png', dpi=300, bbox_inches='tight')
print("💾 Training history plot saved as 'training_history.png'")

# Create salary predictions visualization
plt.figure(figsize=(10, 6))
colors = ['blue', 'green', 'orange', 'red']

for i, education in enumerate(['HighSchool', 'Bachelor', 'Master', 'PhD']):
    # Get data for this education level
    edu_data = data[data['education_level'] == education]

    # Generate predictions across experience range
    exp_range = np.linspace(0, 15, 50)
    predictions = [predict_salary(exp, education) for exp in exp_range]
```

```
    # Plot
    plt.scatter(edu_data['years_experience'], edu_data['salary_usd'],
                color=colors[i], s=80, alpha=0.7, label=f'{education} Actual')
    plt.plot(exp_range, predictions, color=colors[i], linewidth=2,
             label=f'{education} Predicted', linestyle='--')

plt.title('Salary Predictions vs Actual Data')
plt.xlabel('Years of Experience')
plt.ylabel('Salary (USD)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('salary_predictions.png', dpi=300, bbox_inches='tight')
print("💾 Salary predictions plot saved as 'salary_predictions.png'")

# Show file locations
import os
print(f"\n📂 Plots saved in: {os.getcwd()}")
print("   - training_history.png")
print("   - salary_predictions.png")

print("\n☑️ Model training and visualization completed successfully!")
```

(Core TensorFlow model, preprocessing, training, and visualization script.)
☑ Handles:

- Data cleaning and encoding
- Model creation using `tf.keras`
- Training with validation
- Prediction and visualization (`matplotlib`)
- Automatic saving of plots and trained model

🚀 Step 5: Run the Project

☑ Method 1: Using File Explorer

1. Right-click the **Salary_Predictor** folder
2. Choose **"Open in Terminal"**
3. Run:

```
4. tf_env\Scripts\activate
5. python salary_predictor_actual_tf.py
```

☸ Method 2: Using PowerShell

```
cd "C:\Users\johnm\OneDrive\Desktop\Salary_Predictor"
tf_env\Scripts\activate
python salary_predictor_actual_tf.py
```

□ One-Time Setup Script

Save the following as **`setup_project.bat`** in your project folder:

```
@echo off
echo ============================================
echo   Setting up TensorFlow Salary Predictor...
echo ============================================

:: Create and activate virtual environment
py -3.13 -m venv tf_env
call tf_env\Scripts\activate

:: Install required packages
pip install tensorflow pandas numpy matplotlib scikit-learn joblib

echo.
echo ☑ Setup complete! Project ready to use.
echo.
echo To run your project:
echo 1. Navigate to this folder
echo 2. Run: tf_env\Scripts\activate
echo 3. Run: python salary_predictor_actual_tf.py
pause
```

📊 Expected Output Files

File	Description
`salary_predictor_model.h5`	Trained TensorFlow model
`training_history.png`	Training progress graph
`salary_predictions.png`	Predicted vs Actual salaries

File	Description
`scaler.pkl`	Saved Scaler for normalization
`education_mapping.pkl`	Encoded education-level mapping

🔄 Daily Workflow

```
# 1. Navigate to your project folder
cd "C:\Users\johnm\OneDrive\Desktop\Salary_Predictor"

# 2. Activate the environment
tf_env\Scripts\activate

# 3. Run your TensorFlow script
python salary_predictor_actual_tf.py

# 4. (Optional) Launch JupyterLab for exploration
jupyter lab
```

💡 Pro Tips

- Keep the virtual environment activated while working
- Deactivate when finished:
- `deactivate`
- Update packages periodically:
- `pip install --upgrade tensorflow pandas`
- List installed packages:
- `pip list`

🆘 Troubleshooting

If issues occur:

```
# Reinstall dependencies
pip install --force-reinstall tensorflow pandas

# Check Python version
python --version

# Verify environment activation
# Should see (tf_env) before your prompt
```
