Predicting Spatio-temporal temperature variations using Machine learning models

AU2140057 - Sanket Shah, AU2140119 - Ruchil Shah, AU2140165 - Saumya Shah, AU2140168 - Bhavya Jhaveri.

***Abstract*— In this project, we propose a predictive model for temperature forecasting utilizing Convolutional LSTM networks. Temperature forecasting is essential for various applications, including weather prediction and industrial processes. Traditional methods often need help capturing complex temporal dependencies in temperature data. Deep learning techniques, such as Convolutional LSTM networks, offer a promising solution by effectively integrating spatial and temporal information. We present a methodology to preprocess time-series temperature data, construct a Convolutional LSTM architecture, and train the model for temperature prediction. Additionally, we demonstrate the model's performance on validation data and provide visualization of predicted temperature contours.**

This project also focuses on solving the classification problem of distinguishing between velocity and temperature images using machine learning techniques.Our approach involves preprocessing the image data, extracting relevant features, and training classification models. Through extensive experimentation and evaluation, we demonstrate the effectiveness of our approach in accurately categorizing velocity and temperature images.

***Index Terms*— Deep Learning, Transfer Learning, Convolutional Neural Networks, ConvLSTM, Image Classification, Time Series Forecasting, VGG19, Temperature Prediction.**

I. Introduction

TEMPERATURE forecasting and image class- sification are two critical tasks with diverse applications across various fields. In this project, we aim to develop predictive models for temperature forecasting using Convolutional LSTM neural networks and image classification using transfer learning with VGG19.

Temperature prediction is essential for weather forecasting, climate modelling, and industrial process control. Traditional methods struggle with capturing complex temporal dependencies, while deep learning techniques, such as Convolutional LSTM networks offer an effective solution by integrating spatial and temporal dependencies.

On the other hand, image classification is funda- mental in computer vision, classifying velocity and temperature image is a challenging problem with essential applications in numerous domains.

By combining these approaches, our project aims to add dress challenges in both temperature forecasting and image classification, offering a comprehensive solution with potential applications in various domains.

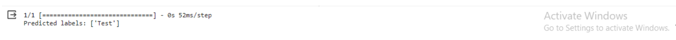
II. Methodology

Temperature Forecasting using Convolutional LSTM Networks:

1) - Preprocessing Time-Series Temperature Data: Start by preprocessing the temperature data to make it suitable for input into the Convolutional LSTM network. This may involve normalization, scaling, and possibly feature engineering to capture relevant temporal patterns.
   - Constructing Convolutional LSTM Architecture: Design a Convolutional LSTM architecture tailored for temperature forecasting. This architecture should effectively integrate spatial and temporal information from the input data.
   - Training the Model: Train the Convolutional LSTM model using the preprocessed temperature data. Utilize techniques such as backpropagation and optimization algorithms to minimize prediction errors and improve model performance.
   - Performance Evaluation: Evaluate the trained model's performance on validation data. This could involve metrics like mean squared error, accuracy, or other appropriate evaluation criteria.
   - Visualization of Predicted Temperature Contours: Visualize the predicted temperature contours to provide insights into the model's forecasting capabilities.

2) *Image Classification using Transfer Learning with VGG19:*
   - Preprocessing Image Data: Prepare the velocity and temperature images for classification. This could involve resizing, normalization, and other preprocessing steps to ensure uniformity and compatibility with the VGG19 architecture.
   - Feature Extraction: Utilize transfer learning with VGG19 to extract relevant features from the image data. This involves leveraging pre-trained layers of VGG19 to capture hierarchical features.
   - Training Classification Models: Train classification models using the extracted features. This could involve techniques such as support vector machines, random forests, or neural networks.
   - Experimentation and Evaluation: Conduct extensive experimentation to evaluate the effectiveness of the classification approach. Measure classification performance using metrics like accuracy, precision, recall, and F1-score.

III. Results



```
1/1 [==============================] - 0s 52ms/step
Predicted labels: ['Test']
```

Fig. 1. Result Image

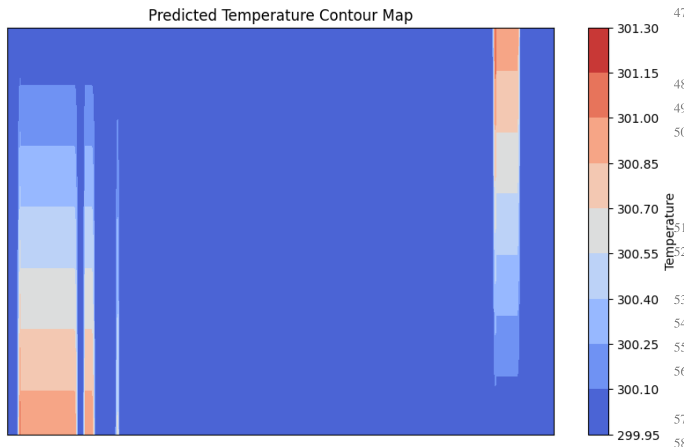


Fig. 2. Result Image

```python
import tensorflow as tf
from keras import layers, models
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG19
import numpy as np
import os

# Path to the directory containing your face images
train_dir = '/content/Data'
test_dir = '/content/Data'

# Create ImageDataGenerator instances for train and test sets
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow images from directories and resize them to (32, 32)
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(32, 32),
        batch_size=32,
        class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(32, 32),
        batch_size=1,
        class_mode='categorical')

# Get the number of classes (individuals)
num_classes = len(train_generator.class_indices)

# Transfer learning using VGG19 as the base model
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the convolutional layers
for layer in base_model.layers:
    layer.trainable = False

# Create a new model on top of the VGG19 base
model_transfer = models.Sequential()

# Add the VGG19 base model (excluding the top layers) to the new model
model_transfer.add(base_model)
model_transfer.add(layers.GlobalAveragePooling2D())
model_transfer.add(layers.Dense(128, activation='relu'))
model_transfer.add(layers.Dense(64, activation='relu'))
model_transfer.add(layers.Dense(num_classes, activation='sigmoid'))  # Sigmoid for multi-label classification

# Compile the transfer learning model
model_transfer.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Binary crossentropy for multi-label classification

# Display the summary of the transfer learning model architecture
model_transfer.summary()

# Change the batch size
batch_size = 8  # Adjust this according to your dataset size

# Train the transfer learning model
history_transfer = model_transfer.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=1,
    validation_data=test_generator,
    validation_steps=test_generator.samples // batch_size
)

# Evaluate the transfer learning model on the test set
test_loss_transfer, test_acc_transfer = model_transfer.evaluate(test_generator)
print(f'Transfer learning test accuracy: {test_acc_transfer}')

from keras.preprocessing import image
import numpy as np

def load_and_preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(32, 32))  # Assuming the target size matches the input size of your model
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Load and preprocess the image
input_image = load_and_preprocess_image('/content/Data/Test_velocity/vel_vect_10s.JPG')

# Make prediction using the transfer learning model
predictions = model_transfer.predict(input_image)

# For single-label classification
if len(train_generator.class_indices) == 1:
    # Decode the prediction
    predicted_class = 'Positive' if predictions[0][0] > 0.5 else 'Negative'
    print("Predicted class:", predicted_class)
else:
    # For multi-label classification
    threshold = 0.5  # Adjust this threshold as needed
    predicted_labels = [label for i, label in enumerate(train_generator.class_indices) if predictions[0][i] > threshold]
    print("Predicted labels:", predicted_labels)
```

Listing 1. Python Code Example

IV. Introduction

This code presents the implementation of a Convolutional LSTM model for time series prediction. The model is trained on temperature data obtained from CSV files containing time information in seconds.

V. Data Loading and Preprocessing

```python
import os
import numpy as np
import pandas as pd
from datetime import datetime

def load_data_with_time(directory):
    data = []
    for filename in os.listdir(directory):
        if filename.endswith(".csv"):
            filepath = os.path.join(directory,
    filename)
            time_seconds = int(filename.split('_')
    [0])
            time = datetime.fromtimestamp(
    time_seconds)
            df = pd.read_csv(filepath)
            df.fillna(df.mean(), inplace=True)
            df['Time'] = time
            data.append(df)
    return pd.concat(data, ignore_index=True)
```

VI. Model Architecture and Training

```python
from keras.models import Sequential
from keras.layers import Conv1D, LSTM, Dense
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam

lookback = 19
n_features = 1

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3,
    activation='relu', input_shape=(lookback,
    n_features)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

early_stopping = EarlyStopping(monitor='val_loss',
    patience=5, verbose=1, mode='min',
    restore_best_weights=True)

history = model.fit(X_train, y_train, epochs=100,
    batch_size=64, validation_split=0.1, callbacks=[
    early_stopping], verbose=1)
```

VII. Validation and Prediction

```python
val_data_directory = "/content/Val_data"
val_raw_data = load_data_with_time(
    val_data_directory)

X_val, y_val = create_samples(val_raw_data, lookback
    )
X_val = X_val.reshape((X_val.shape[0], X_val.shape
    [1], n_features))

y_pred_val = model.predict(X_val)

plt.plot(y_val, label='Actual')
plt.plot(y_pred_val, label='Predicted')
plt.legend()
plt.title('Validation data: Actual vs Predicted
    Temperature')
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.show()

plt.figure(figsize=(10, 6))
contour = plt.contourf(time_steps,
    temperature_values, y_pred_val_2d_reshaped, cmap
    ='coolwarm')
plt.colorbar(contour, label='Temperature')
plt.title('Predicted Temperature Contour Map')
plt.show()
```

So for the classification problem, we have first trained the model and then gave the image for training and also gave a image for testing so it is accurately predicting that this image is from which class.
For the prediction problem, we have first provided the data and then we also trained the data through convo-lstm and then predicted the output for one second ahead of it and also validate it through the validation set where we have provided the right data to test the deviation of it from the original data and then we are converting that predicted output in 2d form and then we are plotting it in the form of contour.

VIII. Discussion

IX. Classification Task

1) **Model Selection and Transfer Learning:** In the classification task, a convolutional neural network (CNN) model is employed. Transfer learning is utilized to leverage the pre-trained VGG19 model as the base architecture. This approach is beneficial when dealing with limited data and helps in extracting relevant features from images effectively.
2) **Data Preprocessing:** Image data generators are used to preprocess the image data by rescaling pixel values to the range [0, 1]. The images are resized to a smaller size (32x32) to reduce computational complexity while preserving essential features.
3) **Model Architecture:** The transfer learning model consists of the VGG19 base model (excluding the top layers), followed by global average pooling and fully connected layers. The activation function used in the dense layers is ReLU, and the final layer employs a sigmoid activation function for multi-label classification.
4) **Training and Evaluation:** The model is trained using the Adam optimizer and binary cross-entropy loss function. Model performance is evaluated using accuracy metrics on both the training and test sets.
5) **Prediction:** Single-label classification is employed when there is only one predicted class, while multi-label classification is used when multiple labels are predicted. A threshold is applied for multi-label classification to determine the predicted labels based on the confidence scores generated by the model.

X. Prediction Task

1) **Model Selection and Architecture:** For the prediction task, a Convolutional LSTM model is utilized. The

model architecture includes a combination of convolutional and LSTM layers, which enables the model to capture spatial and temporal dependencies in sequential data effectively.

2) **Data Preparation:** Time series data is loaded from CSV files, with each file containing time information in seconds. Data preprocessing involves handling missing values by filling them with the mean of the column and converting the time series data into samples suitable for the Convolutional LSTM model.
3) **Model Training:** The Convolutional LSTM model is trained using the Mean Squared Error (MSE) loss function and the Adam optimizer. Early stopping is employed to prevent overfitting and improve generalization by monitoring validation loss and restoring the best weights.
4) **Validation and Prediction:** Validation data from a different directory is loaded to evaluate the model's performance. Predictions are made for the validation data, and actual vs. predicted values are visualized to assess the model's predictive accuracy.

XI. Conclusion

The project addresses temperature forecasting and image classification challenges through innovative methodologies. Comprehensive solutions are developed by leveraging Convolutional LSTM networks for temperature forecasting and transfer learning with VGG19 for image classification.

As validated on validation data, the Convolutional LSTM model effectively integrates spatial and temporal information for accurate temperature predictions. In image classification, transfer learning with VGG19 demonstrates promising results in distinguishing velocity and temperature images.

The combined approaches offer a comprehensive solution applicable to weather prediction, industrial processes, computer vision, and beyond, paving the way for diverse applications.

XII. References

1) Pawar, S., Rahman, S. M., Vaddireddy, H., San, O., Rasheed, A., & Vedula, P. (2019). A deep learning enabler for nonintrusive reduced order modeling of fluid flows. Physics of Fluids, 31(8).

2) Amato, F., Guignard, F., Robert, S., & Kanevski, M. (2020). A novel framework for spatio-temporal prediction of environmental data using deep learning. Scientific Reports, 10(1).

3) Ahmed, S. E., San, O., Rasheed, A., & Iliescu, T. (2021). Nonlinear proper orthogonal decomposition for convection-dominated flows. Physics of Fluids, 33(12).

4) Arun. (2022, January 6). Spatio-Temporal Data Analysis using Deep Learning - Arun - Medium. Medium.