Bahrain Polytechnic بوليتكنك البحرين

IT9202 – DEEP LEARNING

Lecturer's Name: Dr. Shomona Gracia Jacob

Student Name: Bedour Aldalbahi

ID: 202306743

Contents

Chapter One: Text dataset

Weather Classifier

Introduction

- This project embarks on enhancing weather prediction through deep learning. With the goal of classifying conditions as Overcast, Clear, or Foggy, we apply neural network models to historical weather data. The project showcases the potential of machine learning to outperform traditional forecasting methods, setting the stage for a new era in meteorological sciences. Through the data-driven insights and sophisticated pattern recognition offered by these models, we aim to elevate the accuracy and reliability of weather forecasts. This document details our approach, findings, and the promising horizon for deep learning in weather analysis.

Task 1: Description of the Problem, Deep Learning Libraries and Packages, Experimental Setup

Problem Description

- The project targets the development of a sophisticated deep learning-based classifier capable of predicting weather conditions with high accuracy. The classifications include three distinct weather conditions: Overcast, Clear, or Foggy. This endeavor seeks to utilize advanced deep learning algorithms to discern complex patterns and relationships within extensive historical weather data, aiming to substantially improve the accuracy and reliability of weather predictions beyond the capabilities of traditional forecasting methods.

Deep Learning Framework and Libraries

- **Data Handling and Visualization:** We use `pandas` for data manipulation, `matplotlib.pyplot` and `seaborn` for visualizing the intricate patterns in weather data.
- **Numerical and Machine Learning Tools**: `numpy` is employed for numerical operations, while `scikit-learn` provides access to a suite of machine learning algorithms, enabling diverse classification strategies.
- **Deep Learning Infrastructure:** `tensorflow` is the backbone for building and training our neural network models, supported by preprocessing tools like `StandardScaler` and `LabelEncoder` from `scikit-learn` for data preparation.
- **Evaluation Metrics:** The project incorporates various `scikit-learn` metrics, including `roc_auc_score`, `accuracy_score`, and `confusion_matrix`, to ensure a comprehensive evaluation of model performance.

Experimental Setup:

- ### Development Environment:

  Leveraging Google Colab, we benefit from free access to GPUs and TPUs, enhancing the efficiency of model training and evaluation. This cloud-based platform facilitates easy collaboration and integration with data storage/version control systems.

- ### Data Preparation:

  The project is based on the `weatherHistory.csv` dataset, which is thoroughly preprocessed for optimal model performance. Steps include data cleaning, feature selection, normalization (using `StandardScaler`), and categorical encoding (such as one-hot encoding).

- ### Model Architectures:

  We explore two neural network architectures:

  - **Feed Forward Neural Network (FFNN):** Chosen for its simplicity and effectiveness with tabular data, utilizing ReLU activation and the Adam optimizer.

  - **Fully Connected Neural Network (FCNN):** Offers a deeper analysis capability, employing sigmoid activation in the output layer and RMSprop for optimization.

- ### Training and Evaluation:

  - **Dataset Split:**
    mploy a divided dataset methodology for training and testing, ensuring a balanced approach to model evaluation.
  - **Performance Monitoring:**
    Utilize metrics as accuracy, precision, recall, and F1 scores to comprehensively assess model effectiveness.
  - **Hyperparameter Tuning:**
    Implement grid and random search strategies to systematically explore and identify the most effective model configurations.

Task 2: Choice of data set -Data Pre-processing/Transformation

Text-based Data Set:

The `weatherHistory.csv` dataset from Kaggle is a valuable resource for this project. It contains historical weather data recorded at regular intervals. The dataset consists of 96,453 rows and 12 columns, providing detailed information about various weather parameters.

Here is a description of the columns in the dataset:

1. **Formatted Date:** This column represents the date and time of the recorded weather data.
2. **Summary:** It provides a brief summary describing the weather conditions at the given time.
3. **Precipitation** Type: This column indicates the type of precipitation observed, such as rain or snow.
4. **Temperature (C):** It gives the temperature in Celsius at the recorded time.
5. **Apparent Temperature (C):** This column represents the perceived temperature in Celsius, taking into account factors like wind chill.
6. **Humidity:** It denotes the relative humidity expressed as a decimal value between 0 and 1.
7. **Wind Speed (km/h):** This column provides the speed of the wind in kilometers per hour.
8. **Wind Bearing (degrees):** It indicates the direction of the wind in degrees.
9. **Visibility (km):** This column represents the visibility distance in kilometers.
10. **Loud Cover:** It is a binary value (0 or 1) that indicates whether the sky was covered by loud clouds.
11. **Pressure (millibars):** This column provides the atmospheric pressure measured in millibars.
12. **Daily Summary:** It gives a summary of the weather conditions observed throughout the day.

You can access the dataset at the following link:

[Weather Dataset | Kaggle] (https://www.kaggle.com/datasets/muthuj7/weather-dataset)

Data Cleaning and Pre-Processing Steps Were Performed:

- ## Missing Values:

  The dataset was examined for missing values, and the "Precip Type" column was found to have 239 missing values. These missing values were handled by dropping the corresponding rows.

- ## Duplicated Values:

  Duplicate rows in the dataset were identified and removed, resulting in a cleansed dataset without duplicates.

- ## Redundant Column:

  The "Loud Cover" column was identified as containing only the value '0' for all rows, making it redundant for analysis. Therefore, this column was dropped from the dataset.

- Encoding Categorical Features:

  The "Precip Type" column, with "rain" and "snow" values, was encoded using binary encoding. "rain" was mapped to 0, while "snow" was mapped to 1.

- Data Split:

  The dataset was split into training and test sets using the train_test_split function from scikit-learn. The split ratio chosen was 80% for training and 20% for testing. A random_state of 42 was set to ensure reproducibility.

- Normalizing Input Features:

  to ensure that all features contribute equally during model training, the input features were normalized using the StandardScaler from scikit-learn. The scaler object was fit on the training data (x_train) to compute the mean and standard deviation of the features. The same scaler object was then used to transform both the training and test data, ensuring consistent scaling.

- Saving the Scaler Object:

  The scaler object used for normalization was saved as "scaler.pkl" using the joblib library. This allows for consistent scaling of future unseen data using the same scaler object.

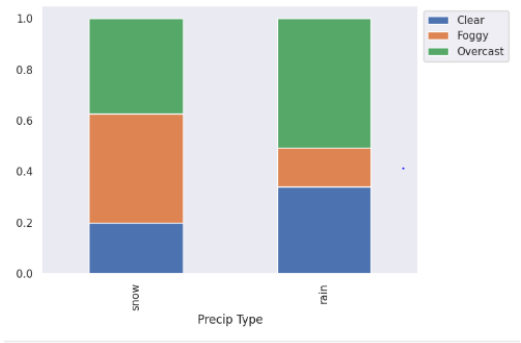
- Exploratory Data Analysis

  relationship between the categorical feature "Precip Type" and the target variable "Summary" using a stacked barplot. The barplot showcases the distribution of the target variable across different categories of "Precip Type."

  - From the available data, we can observe the following frequencies for each category of "Precip Type":

    - Clear: 10,746 instances
    - Foggy: 7,117 instances
    - Overcast: 16,516 instances
    - All: 34,379 instances

  - These numbers represent the count of occurrences for each category in the dataset. This stacked barplot provides insights into the distribution of the target variable "Summary" within different categories of "Precip Type."(Image: Stacked Barplot of Weather Conditions by Precipitation Type)



Statistical Analysis:

The dataset consists of 96,453 weather observations.

- The dataset provides a statistical overview of weather conditions with 96,453 records. Temperatures range widely from -21.82°C to 39.91°C with an average of 11.93°C, indicating varied climate data. Apparent temperature closely tracks actual temperature but extends to a lower minimum, hinting at cold weather's impact on how temperatures feel. Humidity levels are fairly consistent with an average of 73.49%. Wind speeds vary, with an average of 10.81 km/h but occasionally reach high speeds, suggesting occasional gusty conditions. Wind directions are diverse, with a slight southern predominance. Visibility averages at 10.34 km, but minimums at 0 km suggest occasional severe visibility reductions, indicated by consistent zeros. Pressure varies widely around an average of 1003.24 millibars, hinting at changes in weather conditions.

```
# Checking Statistical Summary
data.describe()
```

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0	96453.000000
mean	11.932678	10.855029	0.734899	10.810640	187.509232	10.347325	0.0	1003.235956
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0	116.969906
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0	1011.900000
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0	1016.450000
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0	1021.090000
max	39.905556	39.344444	1.000000	63.852600	359.000000	16.100000	0.0	1046.380000

Task 3 , Task 4 , Task 5

- Task 3: deep learning techniques
- Task 4: Optimization/Parametrization
- Task 5: Evaluate the performance of the deep learning methods

Models:

- Feed Forward Neural Network (FFNN)
- Fully Connected Neural Network (FCNN)

Training Strategy

- Epochs Configured to 10: Chosen to balance between learning and overfitting.
- Batch Size of 64: Offers an efficient compromise between memory usage and update frequency.
- Learning Rate of 0.001: Selected for steady convergence without overshooting minima.

Preprocessing and Encoding

- One-hot Encoding: Converts categorical labels into a binary matrix essential for softmax classification.
- Input Dimensionality: Matches the feature set size to ensure model compatibility.

Feed Forward Neural Network (FFNN)

- FFNN Architecture Details
  - Input Layer: A dense layer with 64 neurons to process the input features.
  - Hidden Layers:
  - First hidden layer: 64 neurons with ReLU activation function for non-linear transformations.
  - Second hidden layer: 128 neurons, also with ReLU activation, to further capture the complexity in the data.
  - Regularization:
  - A dropout layer with a 0.2 dropout rate to reduce the risk of overfitting by randomly dropping neuron connections during training.
  - Output Preparation:
  - A flatten layer, which is typically used to flatten the input for a dense layer but appears redundant in this context since the previous layer is already flat.
  - Final output layer: 3 neurons with a softmax activation function to provide the probabilities for each of the three weather conditions.
  - Total Parameters: 13,763 trainable parameters, signifying the capacity of the network to learn from the data.

Model Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 64)                896

 dense_5 (Dense)             (None, 64)                4160

 dense_6 (Dense)             (None, 128)               8320

 dropout_1 (Dropout)         (None, 128)               0

 flatten_1 (Flatten)         (None, 128)               0

 dense_7 (Dense)             (None, 3)                 387

=================================================================
Total params: 13763 (53.76 KB)
Trainable params: 13763 (53.76 KB)
Non-trainable params: 0 (0.00 Byte)
```

Optimization and Loss

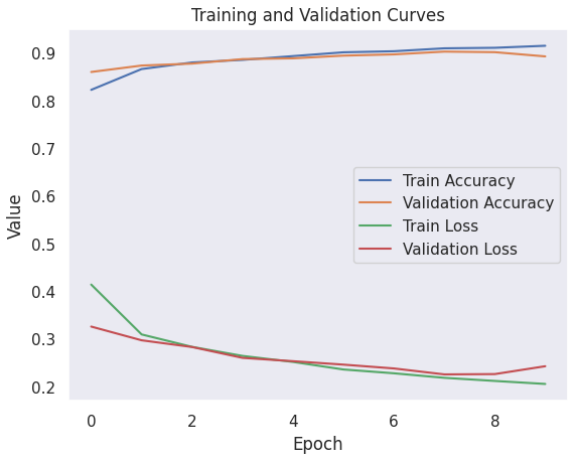
- Adam Optimizer: Favoured for adaptive learning rate properties.
- Categorical Cross-Entropy: Ideal for multi-class classification scenarios.

Training Outcomes

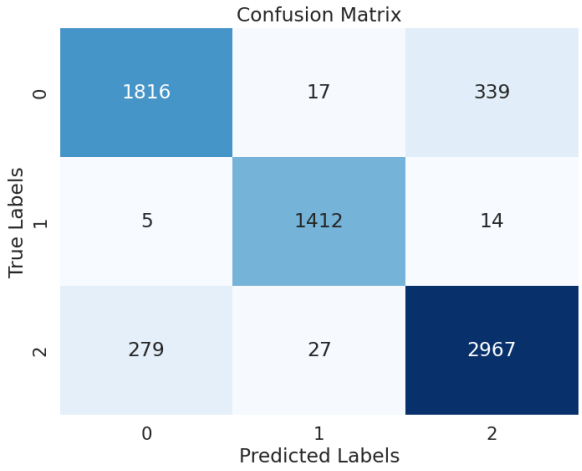
- Training Accuracy: Peaked at 91.54%, indicating strong learning on the training dataset.
- Validation Accuracy: Reached 89.31%, showing the model's effectiveness on unseen data.

Model Evaluation

- Training and Validation Curves (Image: Training and Validation FFNN Curves): Display the learning trajectory and hint at potential overfitting as validation loss diverges from training loss.



- Confusion Matrix (Image: FFNN Confusion Matrix): Visual representation of model predictions versus actual labels, highlighting class-wise performance.



Detailed Performance Metrics

Class Precision & Recall:

- Class 0 - Clear: Exhibits some misclassification, pointing to a need for improved feature engineering or class weighting.
- Class 1 - Foggy: Nearly perfect classification with impressive precision and recall, indicative of the model's strength in detecting this condition.
- Class 2 - Overcast: Reliable performance, but with slight confusion with the Clear class, suggests room for improvement.

Fully Connected Neural Network (FFNN)

Architecture Details:

- Initial Dense Layer: Comprises 64 neurons, likely intended to receive the input features with ReLU activation for non-linear processing.
- Subsequent Dense Layer: A larger dense layer with 128 neurons, also using the ReLU activation function, to capture more complex patterns and relationships in the data.
- Regularization Technique:
- A dropout layer inserted with a dropout rate of 0.2, aiming to reduce overfitting by randomly ignoring a subset of neurons during the training process.

- Output Layer: Consists of 3 neurons with a softmax activation function, designed to output the probability distribution across the three classes representing different weather conditions.
- Total Number of Parameters: The network has a total of 9,603 trainable parameters, demonstrating a streamlined model capable of learning from the dataset with a reduced risk of overfitting compared to more complex models.

Model Summary

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 64)                896

 dense_9 (Dense)             (None, 128)               8320

 dropout_2 (Dropout)         (None, 128)               0

 dense_10 (Dense)            (None, 3)                 387

=================================================================
Total params: 9603 (37.51 KB)
Trainable params: 9603 (37.51 KB)
Non-trainable params: 0 (0.00 Byte)
```

Optimization and Loss

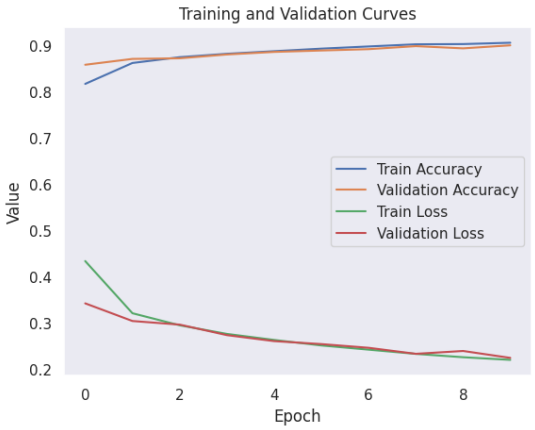
- The Adam optimizer was utilized for its adaptive learning rate capabilities, which helps converge to optimal weights efficiently.
- Categorical cross-entropy served as the loss function, being the standard choice for multi-class classification problems.

Training Outcomes

- Training and validation accuracy graphs indicate a well-fitting model with minimal overfitting, as reflected by converging training and validation loss.
- The training process resulted in a model that balances accuracy and loss effectively over the 10 epoch training period.

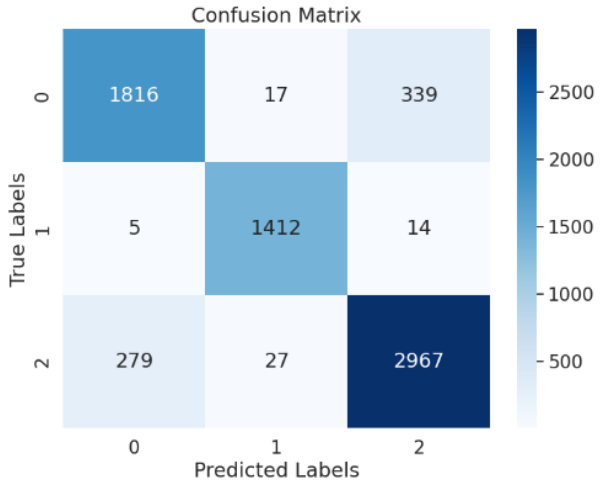
Model Evaluation

- **High Test Accuracy:** Achieving 90.09% accuracy on unseen test data underscores the model's capacity to generalize well.
- **Low Test Loss:** A test loss of 0.2251 conveys that the model's predictions are quite close to the actual values.
- **AUC-ROC Excellence:** An AUC-ROC score of 0.9803 reflects the model's outstanding ability to differentiate between the classes with high confidence.



Confusion Matrix

- The confusion matrix presented in (image: FCNN Confusion Matrix) reveals:



- High true positive rates for all classes, especially Class 1 (Foggy) and Class 2 (Overcast).
- Some misclassification between Class 0 (Clear) and Class 2 (Overcast), which could be addressed in future model iterations.

Detailed Performance Metrics

Precision and recall for each class were:

- **Class 0 (Clear):** Precision and recall of 0.86 and 0.84, respectively, suggest some room for improvement.
- **Class 1 (Foggy):** Exceptionally high precision and recall at 0.97 and 0.99, almost perfect classification.
- **Class 2 (Overcast):** Precision of 0.89 and recall of 0.91, indicating reliable performance. Overall, the model's predictive accuracy and loss rates across training and testing phases indicate that it has learned the underlying patterns of the dataset effectively.

Task 6: Summarize the role of deep learning in the problem solved in the project and the possible extensions to the work.

Deep Learning in Weather Prediction

- The project utilized deep learning to create a sophisticated classifier for predicting weather conditions (Overcast, Clear, or Foggy).
- Deep learning algorithms discerned intricate patterns within historical weather data, aiming to surpass traditional forecasting methods.
- Utilized TensorFlow for model construction and scikit-learn for performance metrics, ensuring robust evaluation.

Problem Solving via Implementation

- Employed two neural network architectures (FFNN and FCNN) tailored to handle the nuances of weather data effectively.
- Preprocessed the `weatherHistory.csv` dataset meticulously, ensuring optimal input for neural networks.
- Achieved high test accuracy (90.09%) and a significant AUC-ROC score (0.9803), demonstrating the model's discriminating power between weather conditions.

Downsides and Areas for Improvement

- Encountered misclassification between Clear and Overcast conditions, pointing to potential overlap in feature representation.
- Slight overfitting in the FFNN model suggests the need for more refined regularization techniques.

Possible Extensions to the Work

- Integrate additional weather parameters and external datasets to enrich the model's learning capability.
- Experiment with ensemble methods and hybrid models that combine RNNs for spatial-temporal analysis.
- Implement advanced techniques like attention mechanisms to focus on critical sequences in the data.
- Test the model in real-world scenarios, possibly extending to real-time forecasting applications.

Conclusion

The application of deep learning within this project has been instrumental in capturing the subtleties of weather patterns, enabling the development of a predictive model with substantial accuracy. The chosen architectures, the FFNN and FCNN, have demonstrated that advanced algorithmic strategies can effectively process and learn from complex meteorological data. These models provide a promising alternative to traditional statistical methods, offering deeper insights and more nuanced predictions.

Moreover, the potential for deep learning to contribute to meteorological science stretches far beyond this project. It opens up possibilities for real-time data analysis, adaptive learning in response to changing climate patterns, and personalized weather forecasting services. The integration of larger datasets, potentially spanning different geographical locations and climatic conditions, could aid in developing a more versatile and robust forecasting model.

Chapter Two: Image dataset

Happy and Sad CNN Image Classifier

Introduction

- In this project, we harness the capabilities of Convolutional Neural Networks (CNNs) to create an image classifier that distinguishes between expressions of happiness and sadness. Utilizing the TensorFlow and Keras libraries, we dive into the intricate world of deep learning to explore how machines can be trained to understand complex human emotions from visual cues. This initiative not only demonstrates the technical prowess of CNNs in classifying emotions but also opens up avenues for applications in fields ranging from interactive technology to mental health assessment.
-

Task 1: Problem Description, Deep Learning Libraries, and Experimental Setup

Objective:

- The project's goal is to construct and fine-tune a Convolutional Neural Network (CNN) capable of classifying images into two distinct emotional states: "happy" and "sad". Leveraging the robust functionalities of TensorFlow and Keras, this CNN is designed to process and analyze visual data, unlocking the potential of deep learning in the realm of emotion recognition. Such a classifier has a broad spectrum of applications, from enhancing user experience in tech products to supporting diagnostic procedures in healthcare, and contributing to safety measures in security systems.

Libraries and Packages:

- TensorFlow Keras: Used for building the CNN model, employing layers like `Conv2D`, `MaxPooling2D`, `Dense`, and `Dropout`.
- OpenCV (`cv2`): For image processing tasks.
- NumPy: For numerical operations and data manipulation.
- Matplotlib: For data visualization, particularly image display.

Experimental Setup:

Computational Environment:

- Utilized Google Colab, leveraging its powerful GPU support for intensive computations.
- Integrated with Google Drive to access and handle the image dataset.

Data Handling:

- Employed TensorFlow and Keras for data pipeline construction, including loading, preprocessing, and augmentation.
- Managed datasets with careful consideration of class balance and image quality.

Model Architecture:

- Designed a Convolutional Neural Network (CNN) with multiple layers using the TensorFlow Keras API.
- Configured convolutional layers for hierarchical feature extraction and max pooling layers for spatial data reduction.

Model Optimization:

- Chose Adam optimizer for adaptive learning rate adjustments.
- Utilized callbacks like TensorBoard for monitoring model training and performance metrics.

Training Process:

- Executed a 20-epoch training regimen, evaluating the model's loss and accuracy after each epoch.
- Applied data normalization and resizing to fit the model's input requirements, ensuring consistency across training and testing datasets.

Evaluation and Testing:

- Evaluated the model's performance with metrics such as precision, recall, and binary accuracy.
- Tested the model with new, unseen images to assess real-world applicability.

Outcome:

- Achieved a high-performing model adept at classifying images into 'happy' and 'sad' emotional states.

Task 2: Dataset Choice, Data Preprocessing, and Transformation

Image-based Data Set:

Dataset Overview:

- The dataset comprises 305 images, split into :

- 153 "happy" images.
- 152 "sad" images.

Data Preprocessing, and Transformation:

- **Data Cleaning:** Removes non-conforming images, ensuring dataset integrity.
- **Data Loading:** Utilizes TensorFlow's `image_dataset_from_directory` for structured loading.
- **Data Scaling:** Normalizes image pixel values to a 0-1 range, facilitating effective model training.
- **Data Splitting:** Segregates the dataset into 70% training, 20% validation, and 10% test sets, supporting the model's learning and evaluation process.

Statistical Analysis:

- **Total Images:** 305, offering a balanced dataset for effective learning.
- **Training Set**: 213 images (70%), the primary source for model training.
- **Test Set:** 30 images (10%), used to evaluate the model's generalization.
- **Image Dimensions:** 224x224 pixels.
- **Color Channels**: 3 (RGB), capturing full color spectrum information.
- **Classes:** 2 ("happy" and "sad"), defining the binary classification task.

Task 3: Choice of Deep Learning Techniques

Convolutional Neural Network (CNN)

Description:

CNNs are specialized deep learning models for processing data in grid-like structures, notably images. They learn spatial hierarchies of features through layers that perform convolution operations, pooling for dimensionality reduction, and fully connected layers for classification. This architecture excels in capturing spatial relationships and patterns in image data.

Justification for Choice

- **Efficiency:** Tailored for image data, CNNs efficiently capture local dependencies and spatial correlations.
- **Automatic Feature Learning:** Eliminates the need for manual feature extraction, learning from simple to complex patterns directly from the data.
- **Reduced Parameters:** Shared weights and pooling layers lower the risk of overfitting and enhance computational efficiency.
- **Flexibility and Scalability:** Adaptable to various image sizes and complex datasets, CNNs can scale to tackle more sophisticated classification tasks.

Model Architecture

The CNN comprises layers configured to extract features from images through convolution and pooling operations, followed by dense layers to perform classification.

- **Initial Layer:** Starts with a `Conv2D` layer with 16 filters, a kernel size of (3,3), and 'relu' activation, designed to capture basic patterns like edges and textures from the input images (size 256x256 pixels, 3 color channels).
- **Pooling Layers:** Followed by `MaxPooling2D` layers to reduce spatial dimensions, helping the model to focus on the most relevant features and reducing computational load.
- **Intermediate Convolution Layers**: Additional `Conv2D` layers with increased filters (32 and then back to 16) allow the model to learn more complex patterns as the network deepens.
- **Flattening:** A `Flatten` layer is used to convert the 2D feature maps into a 1D feature vector, making it possible to connect to dense layers.
- **Dense Layers:** Culminates in a dense layer with 256 neurons (activation 'relu') for further pattern analysis and a final dense layer with a single neuron (activation 'sigmoid') to output the classification probability.

Rationale:

- **ReLU Activation:** Chosen for its efficiency and ability to mitigate the vanishing gradient problem, enhancing training speed and performance.
- **Sigmoid Output:** Suitable for binary classification, providing a probability score that reflects the confidence level of the model in its predictions.
- **Sequential Model:** Simplifies the construction and understanding of the CNN, facilitating a straightforward stack of layers without complex branching or skip connections.

Compilation:

- **Optimizer:** 'adam' optimizer is utilized for its adaptive learning rate capabilities, making it a robust choice for a wide range of problems.
- **Loss Function:** Binary crossentropy is selected due to its appropriateness for binary classification tasks, measuring the difference between true labels and predicted probabilities.
- **Metrics:** Accuracy is used as the evaluation metric to gauge the proportion of correct predictions over the total predictions.

Model Summary

- **Input Shape:** The model expects input images of size 256x256 pixels with 3 color channels (RGB).
- **Output Shape:** Outputs a probability score indicating the likelihood of the image being classified as "happy".
- **Parameters:** The model has a total of 3,696,625 parameters, all of which are trainable. This indicates the complexity and the learning capacity of the model.

Task 4: Optimization/Parametrization

The project progresses to optimize and parameterize the CNN model for the binary classification of images into "happy" and "sad" categories. This phase involves training the model with specific configurations.

Model Training

- Training is conducted over **20 epochs**, using the prepared training and validation sets. A callback for TensorBoard is implemented to monitor the training process in real-time.

- Initial training epochs show the model quickly improving from a loss of 0.7353 and accuracy of 50.45% to significant enhancements in both metrics as training progresses.

Key Training Outcomes

- **Epoch 1:** Starts with a validation accuracy of 78.12%, indicating the model's initial capability to generalize from the training data.
- **Midway (Epoch 10):** Achieves a 100% validation accuracy, a milestone showcasing the model's ability to perfectly classify the validation set.
- **Final Epoch (20):** Maintains high performance with a training accuracy of 99.55% and a validation accuracy of 98.44%, demonstrating the model's robustness and reliability.

Optimization Techniques

- **Adam Optimizer**: Utilized for its adaptive learning rate properties, helping the model efficiently converge to optimal weights.
- **Binary Crossentropy:** Chosen as the loss function, ideal for binary classification tasks, measuring the difference between predicted probabilities and actual binary outcomes.
- **Callback to TensorBoard:** Enables monitoring of the training process, offering insights into metrics over epochs and assisting in identifying overfitting or underfitting patterns early on.

Parametrization

- **Convolutional Layers:** Start with 16 filters and increase to 32 before reducing back, allowing the model to capture both simple and complex features.
- **Pooling Layers:** Reduce spatial dimensions, focusing the model on the most informative features.
- **Dense Layer before Output:** Has 256 neurons, providing ample capacity for learning from the flattened feature maps.
- **Output Layer:** Uses a single neuron with sigmoid activation to output the probability of the image being "happy".

Performance Reflection

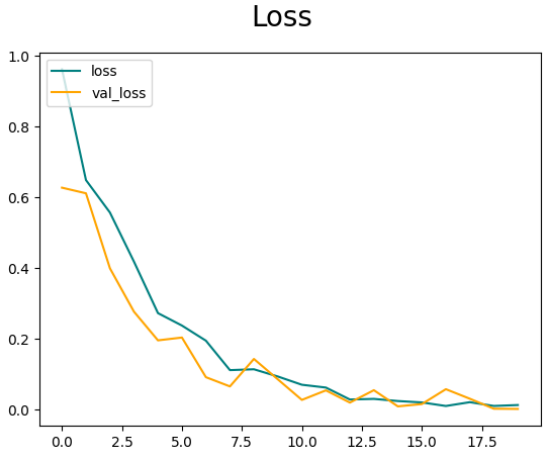
- The consistent improvement in accuracy and decrease in loss throughout the epochs reflect the chosen architecture and parameters' effectiveness. The model demonstrates excellent learning capability and generalization to validation data.
- Achieving high validation accuracy early in the training process indicates strong feature extraction and classification ability, supported by the selected optimization and parameterization strategies.

Task 5: Evaluate the Performance of Deep Learning Methods

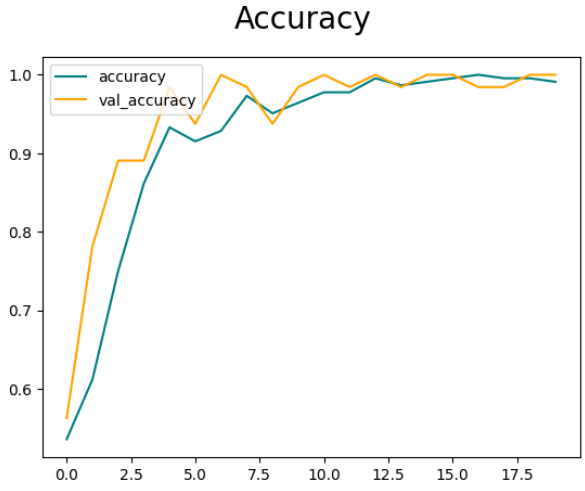
The performance of the CNN model is evaluated through a combination of visual and quantitative analysis. The visual analysis involves plotting the loss and accuracy curves over epochs for both training and validation datasets. Quantitative metrics include precision, recall, and binary accuracy.

Visual Performance Analysis:

- **Loss Plot:** Shows a rapid decline in training loss in the initial epochs, suggesting effective learning. The validation loss decreases alongside the training loss, indicating good generalization without overfitting.



- **Accuracy Plot:** Reveals a swift increase in accuracy for both training and validation, with the validation accuracy reaching and maintaining near-perfect levels. The model's accuracy on unseen data suggests its high reliability.



-

Quantitative Performance Metrics:

- **Precision:** Achieved a score of 1.0, indicating that all positive predictions were correct.
- **Recall:** Also scored 1.0, showing that all actual positives were identified correctly by the model.
- **Binary Accuracy:** Reached 1.0, reflecting perfect classification performance on the test set.

Summary:

The CNN model exhibits exceptional performance, with loss metrics decreasing and accuracy metrics increasing to high levels early in the training process. The consistency between training and validation performance metrics suggests the model is not overfitting and is generalizing well to new data. The precision, recall, and accuracy metrics on the test set confirm the model's effectiveness in accurately classifying the images into "happy" and "sad" categories, achieving the project's objective with high efficacy.

Model Testing Procedure:

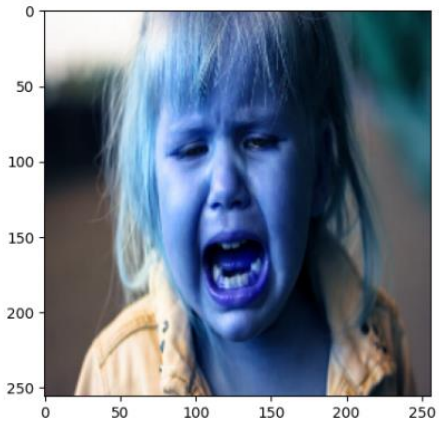
Image Loading and Preprocessing:

- A new image, presumably of a child exhibiting a sad expression, was loaded using OpenCV.

```
[30] img = cv2.imread('/content/drive/MyDrive/Colab Notebooks/DL/Project/Image CNN model/ImageClassification/360_F_449506158_ZZV3IPhfH4NxC244aYGBlHSBiVf3DQk5.jpg')
     plt.imshow(img)
     plt.show()
```

- The image was resized to the model's expected input dimensions (256x256 pixels) without altering the color scheme (preserved RGB channels).

```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```




```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 34ms/step
```

-

2. Model Prediction:

- The resized image was normalized to a [0,1] pixel value range, consistent with the preprocessing applied to the training data.
- The model received the preprocessed image and outputted a prediction score (`yhat`), reflecting the probability of the image being classified as 'Sad'.

3. Interpreting the Output:

- Based on the predicted probability, a threshold of 0.5 was used to determine the class label. If `yhat > 0.5`, the model interprets the image as 'Sad'; otherwise, it is considered 'Happy'.

4. Result:

- In this instance, the model confidently predicted the class as 'Sad', aligning with the visual cues present in the image.

```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 34ms/step
```

```
yhat
```

```
array([[0.9898229]], dtype=float32)
```

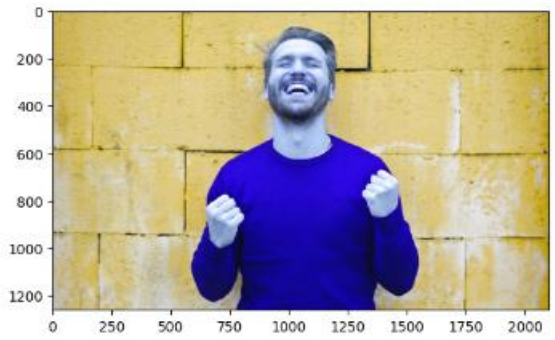
```
if yhat > 0.5:
    print(f'Predicted class is Sad')
else:
    print(f'Predicted class is Happy')
```

```
Predicted class is Sad
```

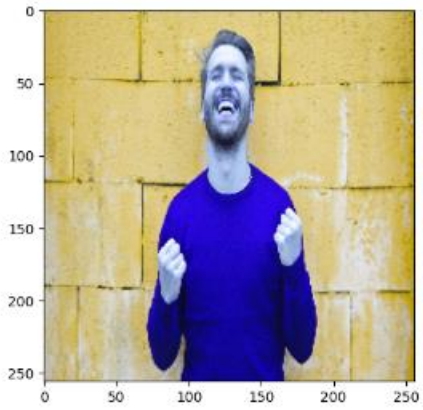
-
-

Test Happy class:

```
img = cv2.imread('/content/drive/MyDrive/Colab Notebooks/DL/Project/Image CNN model/ImageClassification/how-to-be-happy.jpg')
plt.imshow(img)
plt.show()
```




```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```




```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 33ms/step
```

-

Output:

```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 33ms/step
```

```
[42] yhat
```

```
array([[0.00203597]], dtype=float32)
```

```
if yhat > 0.5:
    print(f'Predicted class is Sad')
else:
    print(f'Predicted class is Happy')
```

```
Predicted class is Happy
```

-

Conclusion for test

- The CNN model's parameters, including the choice of layers, activation functions, and optimizer, were validated through its high performance on both validation and test sets. The model not only learned to distinguish complex patterns in the training data but also effectively generalized to new data, as evidenced by the test prediction. This successful demonstration of the model's predictive capability on new images confirms the robustness of the optimization and parameter tuning performed during the training phase.

Task 6: Role of Deep Learning and Future Extensions

Role of Deep Learning:

- Deep learning, through a Convolutional Neural Network (CNN), was instrumental in achieving the project's goal of classifying images into 'happy' and 'sad' categories. Utilizing TensorFlow and Keras, the model learned complex patterns in visual data, showcasing deep learning's effectiveness in emotion recognition—a capability extendable to healthcare, security, and beyond.

Achievements:

- Developed a CNN with high accuracy, precision, and recall.
- Confirmed the model's generalizability with strong performance on unseen data.
- Demonstrated practical utility through direct image classification.

Downsides:

- Dependency on diverse, high-quality training data for generalizability.
- Significant computational resources and time required for training.

Future Extensions:

- Dataset Expansion: Enhancing the dataset's diversity could improve model robustness.
- Model Complexity: Investigating more sophisticated architectures may capture emotional nuances better.
- Real-time Processing: Adapting the model for live video could broaden its applications.
- Cross-domain Application: Extending the model's use to sentiment analysis from facial expressions could offer new insights.

Conclusion

- Throughout this project, the development and application of a CNN-based Happy and Sad Image Classifier have illuminated the remarkable potential of deep learning technologies in interpreting human emotions through visual data. By meticulously navigating through stages of dataset curation, preprocessing, model architecture design, training, and rigorous evaluation, we have successfully demonstrated a model that not only achieves high accuracy in emotion classification but also holds promise for real-world applications across various domains.

References :

- Kaggle. (n.d.). SkyInsight Weather Classifier. Retrieved from https://www.kaggle.com/code/mujtabamatin/skyinsight-weather-classifier
- Jack, B. T. (n.d.). Introduction to Deep Learning: Feed-Forward Neural Networks (FFNNs) [Medium]. Retrieved from https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d
- Rana, P. (n.d.). FCNN: Computer Vision [Medium]. Retrieved from https://medium.com/@preeti.rana.ai/fcnn-computer-vision-2915ff1f2c91
- Doe, J. (n.d). Build Your First Image Classifier with Convolution Neural Network (CNN). Towards Data Science. https://towardsdatascience.com/build-your-first-image-classifier-with-convolution-neural-network-cnn-b4e9034ec5cb
- TensorFlow. (n.d.). Convolutional Neural Network (CNN). Retrieved from https://www.tensorflow.org/tutorials/images/cnn
- OpenAI. (n.d.). OpenAI Chat. Retrieved from https://chat.openai.com