VISION AI Intern Assignment

Table of Contents

Exercise 1 – Image classification

1. Introduction

a. Overview

This problem aims to classify animal images, specifically distinguishing between cats and dogs from the input images. To solve this problem, the model uses MobileNetV2 as a pre-trained model with a deep neural network. This model leverages deep learning techniques for image classification to achieve high accuracy and minimize errors during prediction.

b. Settings

IDE: Kaggle (GPU T4 x2)

Language: Python

2. Pipeline of the assignment

a. Problem statement

- *Input*: The input images have a size of `224x224` with 3 color channels (RGB).
- *Output*: The model classifies the images into two categories: cats and dogs.

b. Method Used

The method uses the **MobileNetV2** model, which has been pre-trained on the **ImageNet** dataset. I do not fine-tune the MobileNetV2 weights. Instead, I

add layers such as `GlobalAveragePooling2D, Dropout`, and `Dense` at the end of the model for binary classification (cat or dog).

- `GlobalAveragePooling2D`: Collapses each feature map into a single value, summarizing spatial information and drastically reducing the number of parameters compared to a Flatten + Dense approach.
- `Dropout`: Randomly zeros out a fraction of the activations at each update, which helps prevent overfitting in the new layers.
- `Dense`: Outputs a probability for the "dog" class (with "cat" = 1 – p), making it ideal for binary classification.

c. Dataset

The Cat and Dog Classification dataset is a standard computer vision dataset that involves classifying photos as either containing a dog or a cat.

This dataset is provided as a subset of photos from a much larger dataset of approximately 25 thousands. The training images are divided equally between cat and dog images.

The dataset is divided into training, validation, and testing sets. The **ratio** of training to testing data is **8:2**, where 80% of the data is used for training and validation, and 20% is kept for testing. Specifically:

- **Training and Validation Split**: The training set is further split into training (80%) and validation (20%) subsets using the `validation_split` argument in the `ImageDataGenerator`. This ensures that 20% of the training data is used for validation during training.
- **Test Set**: The test set is provided separately and remains unchanged for final evaluation.

d. Experiment

i. Data preprocessing

The data is preprocessed using the ImageDataGenerator to normalize the pixel values to the range [0, 1]. Augmentation techniques such as horizontal flipping are also used during training to increase the diversity of the dataset.

ii. Training process

- The model is trained for 5 epochs with a batch size of 64.
- EarlyStopping is used to halt training if there is no improvement in validation loss for 2 consecutive epochs.

```
early_stopping =
EarlyStopping(monitor='val_loss',
patience=2, restore_best_weights=True)
```

- Using Adam as the optimizer with learning_rate=0.001.

```
model.compile(optimizer=Adam(learning_rate=
0.0001), loss='binary_crossentropy',
metrics=['accuracy'])
```

e. Evaluation

i. Metrics

The evaluation metrics used for this task include:

- **Accuracy**: The model's accuracy on the test set.
- **Precision, Recall, F1-Score**: Classification metrics to evaluate the model's performance in correctly classifying both classes.

ii. Experimental Results

After training, the model achieved an accuracy of **97.7%** on the test set. The classification metrics such as precision, recall, and F1-score are also high, with values around **0.98** for both cat and dog classes.

Test classification results is presented:

	precision	recall	f1-score	support
0	0.98	0.97	0.98	2500
1	0.97	0.98	0.98	2500
accuracy			0.98	5000
macro avg	0.98	0.98	0.98	5000
weighted avg	0.98	0.98	0.98	5000

3. Conclusion

a. Advantages

- The model uses MobileNetV2, which delivers good performance with low computational resources due to its lightweight yet powerful architecture.
- High accuracy and classification metrics indicate that the model works stably and effectively.
- EarlyStopping ensures that training is stopped at the right time, preventing overfitting.

b. Disadvantages

- The lack of fine-tuning for MobileNetV2 may prevent the model from fully leveraging specific features of the dataset.
- The model could be improved further by training on a larger dataset or increasing the number of epochs.

Exercise 2 – Text to speech

1. **Introduction to TTS (Text-to-Speech)**

Text-to-Speech (TTS) is a technology that converts text into speech, allowing users to listen to the content of the text rather than read it. The applications of TTS are vast, ranging from reading books, documents, and websites to providing support for people with disabilities or those who are unable to read visually.

2. **Pipeline of the Vietnamese TTS Model**

To build a TTS model for Vietnamese, the pipeline can be divided into the following main steps:

a. **Text Preprocessing**

- **Text normalization**
  - Purpose: Vietnamese has many special terms such as punctuation marks, non-standard Unicode characters, abbreviations, or inconsistent use of uppercase and lowercase letters. Normalizing the text helps make the input data uniform and easier for the model to process.
  - How:
    - Remove unnecessary punctuation: Unnecessary punctuation marks can create noise during model training.
    - Normalize abbreviations: Vietnamese has many abbreviations (e.g., "TPHCM" → "Ho Chi Minh City"). Normalizing these abbreviations helps the model correctly understand common phrases.
- **Tokenization**
  - Purpose Tokenization is the process of dividing text into smaller units (words or phrases). In Vietnamese, there are cases where words are not separated by spaces, for example in compound words or multi-syllable words.
  - How:
    - Use libraries such as VnCoreNLP to help split Vietnamese sentences into words or phrases while maintaining the meaning.
    - For example, "Tôi đi học" would be tokenized into three words: ["Tôi", "đi", "học"]. However, for words like "mái tóc" or "người Việt Nam," the model will accurately determine the boundaries between words.
- **Phoneme Mapping**

- Purpose: A phoneme is the basic sound unit of a language. Converting text into phonemes helps the model understand and pronounce words, phrases, and sentences accurately.
- How:
  - Convert words into phonemes: Each Vietnamese word will be converted into a sequence of phonemes. For example, the word "học" could be converted into the phonemes [h, ɔ, k]. These phonemes help the model recognize how to pronounce the word correctly.
  - Use tools such as VnPhonemizer: a popular library that helps convert Vietnamese text into phoneme sequences. This helps to ensure that the model can pronounce words correctly, especially when there are words that sound the same but have different meanings.

b. Text to Features

Before the model can pronounce text, we need to convert the text into audio features that the model can use. For Vietnamese, the model needs to capture pitch variations (tones) for each syllable to create a natural-sounding voice.

- **Phoneme Representation**: This is a crucial step in TTS, as each word in the text must be converted into a series of phonemes—these are not letters, but the basic sounds that are pronounced.
  - Converting text into phonemes helps the model understand "what to pronounce", rather than just learning to read the text.
  - In Vietnamese, phonemes also need to consider the tones (flat, sharp, heavy, questioning, rising, and falling).
  - Tools like VnPhonemizer can be used for this conversion.
- **Converting to Mel Spectrogram**: The TTS model will need audio features such as Mel-spectrograms for training.
  - It is a visual representation of sound.
  - The sound signals are analyzed into corresponding frequency waves.
  - It is a time-frequency representation - showing the "structure of sound" but not the actual sound yet.

c. Models for Generating Mel Spectrograms from Text

We can use popular deep learning models for TTS, such as:

- **Tacotron 2**: A well-known TTS model, consisting of an encoder-decoder network to convert text into Mel-spectrograms, and a WaveNet or vocoder network to convert Mel-spectrograms into sound waves.
- **FastSpeech**: An improvement over Tacotron, reducing latency in sound generation and can be more efficient with features such as self-attention.

d. Converting Mel-Spectrograms to Sound (Vocoding)

Once the Mel-spectrogram is generated, the next step is to convert it into natural sound, turning it into actual audio. A vocoder will "recreate" the waveform from the Mel-spectrogram, producing a `.wav` audio file. To do this, we can use a vocoder model such as:

- WaveNet: A deep generative network model that can convert Mel-spectrograms into sound waves.
- HiFi-GAN: An efficient vocoder model that can quickly and accurately convert Mel-spectrograms into sound.

3. Algorithms and Models

From my research, there are two common approaches to solve this problem:

- **Tacotron 2 + WaveNet:**
  - Advantages:
    - Natural and smooth voice quality.
    - WaveNet generates realistic sound with rich audio details.
  - Disadvantages:
    - Slow processing: WaveNet is an autoregressive model, that's why it generates audio sample by sample, which makes real-time processing slow.
    - Requires high resources: Training and inference are resource-intensive, requiring a lot of GPU power.
- **FastSpeech + HiFi-GAN:**
  - Advantages:
    - Fast: FastSpeech is a non-autoregressive model, generating the entire Mel-spectrogram at once, making it suitable for real-time applications.
    - HiFi-GAN generates fast and natural-sounding audio of high quality.
    - Easier to train: FastSpeech converges faster and is simpler compared to Tacotron 2.
  - Disadvantages:
    - If training is not done well, the speech may sound less natural than Tacotron 2.

Therefore, I chose **FastSpeech + HiFi-GAN** because:

- Faster audio generation, making it easier to implement in real-time.
- The speech quality is still very good, suitable for building a workable Vietnamese TTS system.
- It is a modern model with plenty of documentation and is easier to customize.

4. Possible Issues and Solutions

a. Pronunciation Issues in Vietnamese

Vietnamese has many phonemes and special vocabulary, which can make it difficult for the model to distinguish between words with similar sounds but different meanings (“bà” and “ba”). To solve this, we need to build a comprehensive and accurate Vietnamese phoneme dictionary.

b. Unnatural Speech Quality

A common problem in TTS models is that the generated voice sounds unnatural or robotic. To address this, we can improve the vocoder model to produce higher-quality and more natural-sounding audio.

REFERENCES

[1] VnCoreNLP
[2] VnPhonemizer
[3] Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions
[4] WaveNet
[5] FastSpeech
[6] HiFi-GAN