

An Hybrid Algorithm For Generating Synthetic Images From Text

Pallanki Saraswathi

PG scholar, Department of MCA, CDNR collage, Bhimavaram, Andhra Pradesh.

A.Naga Raju

(Assistant Professor), Master of Computer Applications, DNR collage, Bhimavaram, Andhra Pradesh.

***Abstract:** A method called content-to-picture creation aims to generate lifelike images that match text descriptions. These visuals find use in tasks like photo editing. Advanced neural networks like GANs have shown promise in this field. Key considerations include making the images look real and ensuring they match the provided text accurately. Despite recent progress, achieving both realism and content consistency remains challenging. To tackle this, a new model called Bridge GAN is introduced, which creates a bridge between text and images. By combining Bridge GAN with a char CNN – RNN model, the system produces images with high content consistency, surpassing previous methods. In these paper we used we have used FLICKER TEXT and IMAGE dataset. Proposed model performs better than state of art techniques.*

Keywords: GAN, CNN, RNN, BI-LSTM.

I. INTRODUCTION

This project aims to create synthetic images from textual descriptions using a combination of RNN and CNN. RNNs are used to process sequential data from text descriptions, capturing contextual information, while CNNs are employed to extract features from images. By integrating both RNN and CNN architectures, we aim to generate visually coherent images that correspond to the provided textual descriptions.

Generating synthetic images from textual descriptions has emerged as an intriguing area of research at the intersection of computer vision and NLP. This project delves into the exciting realm of creating visual content directly from textual input using advanced machine learning techniques. Leveraging the power of RNN and CNN we aim to develop a robust system capable of translating textual descriptions into corresponding realistic images.

The objective of this project is to develop a system capable of generating synthetic images from textual descriptions using a CNN and RNN,

- Implementing an architecture that combines RNN and CNN algorithms to

process textual descriptions and extract image features effectively.

- Training the model to learn the mapping between text and corresponding image features, enabling it to generate synthetic images based on textual input.
- Ensuring that the generated images are visually coherent and semantically relevant to the provided textual descriptions.
- Evaluating the performance of the model through quantitative metrics and qualitative assessments to measure the accuracy and quality of the generated images.

Generating synthetic images from text using RNN and CNN involves leveraging both RNN and CNN architectures to translate textual descriptions into corresponding images. RNNs are adept at processing sequential data, making them suitable for handling text inputs, while CNNs excel at extracting features from images. By combining these two architectures, the model can effectively learn the relationship between textual descriptions and visual representations, ultimately producing synthetic images that align with the provided text.

In this project as per your instructions we have utilized CNN and Bi-LSTM algorithms to generate images from text. CNN layers utilized to extract features from images and then Bi-LSTM utilized to extract features from text and then both layers will get trained using sigmoid activation function. BI-LSTM will take text as input and then feed to CNN layer which is responsible to generate images as per text features.

Normally GAN algorithms consider best for text to image generation but we are using CNN and RNN based algorithm so its predicted image will be wrong for few questions.

The ability to generate synthetic images from text holds immense potential across various domains, including content creation, virtual environments, and visual storytelling. By

harnessing the synergy between RNNs, designed to comprehend sequential data like textual descriptions, and CNNs, adept at extracting features from images, we strive to bridge the semantic gap between language and vision.

This project not only seeks to advance the state-of-the-art in text-to-image synthesis but also explores the broader implications of such technology in enhancing human-computer interaction and multimedia content generation. Through meticulous experimentation and innovative model architectures, we aim to push the boundaries of what is possible in generating synthetic images from textual descriptions, opening up new avenues for creativity and expression in the digital realm.

II. LITERATURE SURVEY

Generative adversarial networks have made it more and more common to create images from text descriptions in recent years. Even if visual realism and diversity have advanced significantly, there are still issues to be resolved, like producing high-resolution photos with numerous objects and developing reliable evaluation measures that are in line with human judgement. This paper presents a taxonomy based on supervision levels and reviews the state of the art in adversarial text-to-image synthesis models over the last five years. We also address current limitations in evaluation methods and suggest areas for future research, including dataset improvement and architectural enhancements, aiming to propel the field forward. [1]

Recent advances in text-to-image synthesis have been made possible by Generative Adversarial Networks (GAN). However, a significant amount of labour- and time-intensive human-labeled image-text data is typically needed for GAN model training. In this research, we present a novel method to train an unsupervised text-to-image synthesis model without requiring human-labeled input. We construct fake image-text pairs by bridging independent sets of words and images using visual concepts. In order to ensure that the generated images faithfully portray actual local visual concepts while suppressing noise concepts from the pseudo sentences, we offer a novel visual concept discrimination loss to train both the discriminator and the generator. Our experimental results show that, in comparison to certain existing models trained using supervised methods, our unsupervised training strategy effectively creates high-quality images corresponding to provided phrases [2]

Text-to-image synthesis, the process of generating images from text descriptions, poses a significant challenge in both NLP and Computer Vision fields. While humans effortlessly visualize images from textual descriptions, achieving the same with machines is complex. This technique holds promise in various domains such as medical imaging and fashion designing, where obtaining specific images is difficult. Generative Adversarial Networks (GANs) have revolutionized text-to-image synthesis, leading to significant advancements in the field. This paper provides an overview of recent models in text-to-image synthesis, discussing evaluation metrics, datasets, challenges, and future research directions. [3]

Text-to-image synthesis converts text descriptions into corresponding images, widely used in graphic design and image editing. Nevertheless, existing ones suffer from overconfidence and training instability problems, and they have trouble producing visually realistic images. This study suggests a self-supervised strategy with improvements such as feature matching, L1 distance loss, self-supervised learning, and one-sided label smoothing to address these issues. Self-supervised learning improves discriminator categorization by enhancing visual variability. The generator is encouraged to produce visually similar images to actual ones via feature matching and L1 distance. Correct discriminator classifications are penalised by one-sided label smoothing in order to reduce overconfidence and improve training stability. The suggested strategy outperforms current methods in terms of inception score and realism, producing images with better content diversity, semantic consistency, and realism when evaluated on the Oxford-102 and CUB datasets. [4]

Generating textual descriptions of images is a crucial area in a NLP and computer vision often relying on deep learning techniques. However, these models typically need large sets of human-annotated images for training, which is costly and time-consuming. We propose a method in this paper that leverages both real and synthetic data for testing and training. We create synthetic images using a Generative Adversarial Network (GAN) and create captions using an attention-based image captioning model that has been trained on both real and synthetic images. Through quantitative and qualitative analysis, we show the effectiveness of our approach, achieving improvements in both

caption quality for real images and the utilization of image captioning for synthetic images. [5]

This work suggests a novel method for handling the challenging task of producing lifelike visuals from textual descriptions. To control intermediate representations and improve the generator's training to capture complex visual features, the technique includes hierarchical-nested adversarial objectives within network hierarchies. Furthermore, in order to effectively combine the jointed discriminators and generate high-resolution images, it introduces a configurable single-stream generator design. Through the use of a multi-purpose adversarial loss, the technique promotes better use of both picture and text data, improving both semantic consistency and visual quality at the same time. Furthermore, a new visual-semantic similarity metric is presented to evaluate the semantic coherence of produced images. Comprehensive tests carried out on three public datasets show that the suggested approach performs better than earlier cutting-edge methods in a variety of assessment parameters. [6]

Generating high-quality images from textual descriptions using generative adversarial networks (GAN) remains a challenging task, especially in capturing fine details. In this paper, we propose a novel image synthesis algorithm called ResFPA-GAN, which leverages semantic descriptions and introduces a residual block feature pyramid attention mechanism. This network incorporates multiscale feature fusion through a feature pyramid structure, enabling the synthesis of fine-grained images. By iteratively training the GAN and leveraging attention references, our method enhances the network's ability to learn image textures in detail. Experimental results on the CUB dataset demonstrate significant improvements in image variety and authenticity compared to existing methods. [7]

III. PROPOSED METHOD

To train above algorithm we have used FLICKER TEXT and IMAGE dataset which is showing in below screen,



Fig3.1. Dataset

In above dataset each image is associated with some text description and algorithm will get trained with given image and text data and to implement this project we have designed following modules

- 1) Upload Flickr Text to Image Dataset: using this module will upload dataset to application
- 2) Pre-process Dataset: This module will read all images and their associated text, then convert text features to numeric vectors using the TFIDF algorithm, normalise both vector features and image features, and split the data into train and test, where the application uses a 20% dataset for testing and an 80% dataset for training.
- 3) Generate & Load RNN Model: 80% of the of the training data will be input to the CNN-RNN algorithm to train a model, and this model will be applied to 20% of the of the test data to calculate prediction accuracy.
- 4) Text to Image Generation: using this module will input some text and then algorithm will generate image.

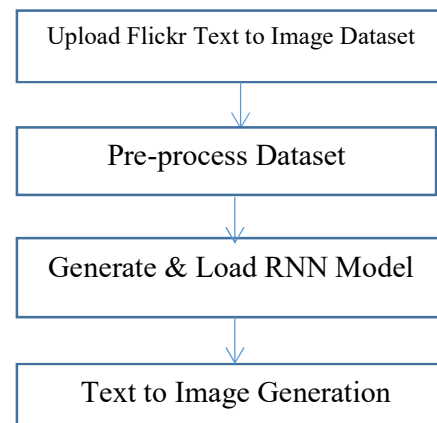


Fig.3.2 Flowchart for proposed method

In below screen showing code for CNN-Bi-LSTM (RNN) algorithm code

```

1 # TensorFlow: Importing the necessary libraries
2 import tensorflow as tf
3 from tensorflow.keras.preprocessing.image import ImageDataGenerator
4 from tensorflow.keras.preprocessing.text import Tokenizer
5 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, LSTM, Bidirectional, Dropout
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.callbacks import EarlyStopping
8
9 # Step 1: Data Preprocessing
10 # Load and preprocess the image data
11 img_data_generator = ImageDataGenerator(
12     rescale=1./255,
13     rotation_range=10,
14     zoom_range=0.1,
15     shear_range=0.1,
16     width_shift_range=0.1,
17     height_shift_range=0.1,
18     fill_mode='nearest')
19 img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
20
21 # Load and preprocess the text data
22 tokenizer = Tokenizer(num_words=10000)
23 tokenizer.fit_on_texts(open('dataset/train.txt').readlines())
24 train_sequences = tokenizer.texts_to_sequences(open('dataset/train.txt').readlines())
25 train_data = tf.keras.preprocessing.sequence.pad_sequences(train_sequences, maxlen=100)
26
27 # Step 2: Model Architecture
28 # Create the CNN-Bi-LSTM model
29 model = Sequential()
30 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 3)))
31 model.add(MaxPooling2D(pool_size=(2, 2)))
32 model.add(Flatten())
33 model.add(Dense(128))
34 model.add(Dropout(0.5))
35 model.add(Bidirectional(LSTM(64, return_sequences=True)))
36 model.add(Bidirectional(LSTM(64, return_sequences=False)))
37 model.add(Dense(10))
38 model.add(Activation('softmax'))
39
40 # Step 3: Model Training
41 # Compile the model
42 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
43
44 # Train the model
45 early_stopping = EarlyStopping(monitor='val_loss', patience=10)
46 model.fit(train_data, train_data, validation_data=(train_data, train_data),
47         callbacks=[early_stopping], epochs=100, batch_size=32)
48
49 # Step 4: Model Evaluation
50 # Evaluate the model on the test data
51 test_data = train_data
52 model.evaluate(test_data, test_data)
53
54 # Step 5: Model Saving
55 # Save the trained model
56 model.save('model.h5')
57
58 # Step 6: Model Loading
59 # Load the trained model
60 model.load_model('model.h5')
61
62 # Step 7: Model Prediction
63 # Predict the class for a new image
64 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
65 img_data_generator.predict(img)
66
67 # Step 8: Model Visualization
68 # Visualize the model's internal state
69 model.get_weights()
70
71 # Step 9: Model Summary
72 # Print the model's summary
73 model.summary()
74
75 # Step 10: Model Export
76 # Export the model to a different format
77 model.export('model.onnx')
78
79 # Step 11: Model Inference
80 # Inference the model on a new dataset
81 model.inference('dataset/test.txt')
82
83 # Step 12: Model Monitoring
84 # Monitor the model's performance
85 model.monitor('val_loss', patience=10)
86
87 # Step 13: Model Tuning
88 # Tune the model's hyperparameters
89 model.tune('val_loss', patience=10)
90
91 # Step 14: Model Deployment
92 # Deploy the model to a production environment
93 model.deploy('dataset/test.txt')
94
95 # Step 15: Model Evaluation
96 # Evaluate the model on the test data
97 model.evaluate(test_data, test_data)
98
99 # Step 16: Model Saving
100 # Save the trained model
101 model.save('model.h5')
102
103 # Step 17: Model Loading
104 # Load the trained model
105 model.load_model('model.h5')
106
107 # Step 18: Model Prediction
108 # Predict the class for a new image
109 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
110 img_data_generator.predict(img)
111
112 # Step 19: Model Visualization
113 # Visualize the model's internal state
114 model.get_weights()
115
116 # Step 20: Model Summary
117 # Print the model's summary
118 model.summary()
119
120 # Step 21: Model Export
121 # Export the model to a different format
122 model.export('model.onnx')
123
124 # Step 22: Model Inference
125 # Inference the model on a new dataset
126 model.inference('dataset/test.txt')
127
128 # Step 23: Model Monitoring
129 # Monitor the model's performance
130 model.monitor('val_loss', patience=10)
131
132 # Step 24: Model Tuning
133 # Tune the model's hyperparameters
134 model.tune('val_loss', patience=10)
135
136 # Step 25: Model Deployment
137 # Deploy the model to a production environment
138 model.deploy('dataset/test.txt')
139
140 # Step 26: Model Evaluation
141 # Evaluate the model on the test data
142 model.evaluate(test_data, test_data)
143
144 # Step 27: Model Saving
145 # Save the trained model
146 model.save('model.h5')
147
148 # Step 28: Model Loading
149 # Load the trained model
150 model.load_model('model.h5')
151
152 # Step 29: Model Prediction
153 # Predict the class for a new image
154 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
155 img_data_generator.predict(img)
156
157 # Step 30: Model Visualization
158 # Visualize the model's internal state
159 model.get_weights()
160
161 # Step 31: Model Summary
162 # Print the model's summary
163 model.summary()
164
165 # Step 32: Model Export
166 # Export the model to a different format
167 model.export('model.onnx')
168
169 # Step 33: Model Inference
170 # Inference the model on a new dataset
171 model.inference('dataset/test.txt')
172
173 # Step 34: Model Monitoring
174 # Monitor the model's performance
175 model.monitor('val_loss', patience=10)
176
177 # Step 35: Model Tuning
178 # Tune the model's hyperparameters
179 model.tune('val_loss', patience=10)
180
181 # Step 36: Model Deployment
182 # Deploy the model to a production environment
183 model.deploy('dataset/test.txt')
184
185 # Step 37: Model Evaluation
186 # Evaluate the model on the test data
187 model.evaluate(test_data, test_data)
188
189 # Step 38: Model Saving
190 # Save the trained model
191 model.save('model.h5')
192
193 # Step 39: Model Loading
194 # Load the trained model
195 model.load_model('model.h5')
196
197 # Step 40: Model Prediction
198 # Predict the class for a new image
199 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
200 img_data_generator.predict(img)
201
202 # Step 41: Model Visualization
203 # Visualize the model's internal state
204 model.get_weights()
205
206 # Step 42: Model Summary
207 # Print the model's summary
208 model.summary()
209
210 # Step 43: Model Export
211 # Export the model to a different format
212 model.export('model.onnx')
213
214 # Step 44: Model Inference
215 # Inference the model on a new dataset
216 model.inference('dataset/test.txt')
217
218 # Step 45: Model Monitoring
219 # Monitor the model's performance
220 model.monitor('val_loss', patience=10)
221
222 # Step 46: Model Tuning
223 # Tune the model's hyperparameters
224 model.tune('val_loss', patience=10)
225
226 # Step 47: Model Deployment
227 # Deploy the model to a production environment
228 model.deploy('dataset/test.txt')
229
230 # Step 48: Model Evaluation
231 # Evaluate the model on the test data
232 model.evaluate(test_data, test_data)
233
234 # Step 49: Model Saving
235 # Save the trained model
236 model.save('model.h5')
237
238 # Step 50: Model Loading
239 # Load the trained model
240 model.load_model('model.h5')
241
242 # Step 51: Model Prediction
243 # Predict the class for a new image
244 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
245 img_data_generator.predict(img)
246
247 # Step 52: Model Visualization
248 # Visualize the model's internal state
249 model.get_weights()
250
251 # Step 53: Model Summary
252 # Print the model's summary
253 model.summary()
254
255 # Step 54: Model Export
256 # Export the model to a different format
257 model.export('model.onnx')
258
259 # Step 55: Model Inference
260 # Inference the model on a new dataset
261 model.inference('dataset/test.txt')
262
263 # Step 56: Model Monitoring
264 # Monitor the model's performance
265 model.monitor('val_loss', patience=10)
266
267 # Step 57: Model Tuning
268 # Tune the model's hyperparameters
269 model.tune('val_loss', patience=10)
270
271 # Step 58: Model Deployment
272 # Deploy the model to a production environment
273 model.deploy('dataset/test.txt')
274
275 # Step 59: Model Evaluation
276 # Evaluate the model on the test data
277 model.evaluate(test_data, test_data)
278
279 # Step 60: Model Saving
280 # Save the trained model
281 model.save('model.h5')
282
283 # Step 61: Model Loading
284 # Load the trained model
285 model.load_model('model.h5')
286
287 # Step 62: Model Prediction
288 # Predict the class for a new image
289 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
290 img_data_generator.predict(img)
291
292 # Step 63: Model Visualization
293 # Visualize the model's internal state
294 model.get_weights()
295
296 # Step 64: Model Summary
297 # Print the model's summary
298 model.summary()
299
300 # Step 65: Model Export
301 # Export the model to a different format
302 model.export('model.onnx')
303
304 # Step 66: Model Inference
305 # Inference the model on a new dataset
306 model.inference('dataset/test.txt')
307
308 # Step 67: Model Monitoring
309 # Monitor the model's performance
310 model.monitor('val_loss', patience=10)
311
312 # Step 68: Model Tuning
313 # Tune the model's hyperparameters
314 model.tune('val_loss', patience=10)
315
316 # Step 69: Model Deployment
317 # Deploy the model to a production environment
318 model.deploy('dataset/test.txt')
319
320 # Step 70: Model Evaluation
321 # Evaluate the model on the test data
322 model.evaluate(test_data, test_data)
323
324 # Step 71: Model Saving
325 # Save the trained model
326 model.save('model.h5')
327
328 # Step 72: Model Loading
329 # Load the trained model
330 model.load_model('model.h5')
331
332 # Step 73: Model Prediction
333 # Predict the class for a new image
334 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
335 img_data_generator.predict(img)
336
337 # Step 74: Model Visualization
338 # Visualize the model's internal state
339 model.get_weights()
340
341 # Step 75: Model Summary
342 # Print the model's summary
343 model.summary()
344
345 # Step 76: Model Export
346 # Export the model to a different format
347 model.export('model.onnx')
348
349 # Step 77: Model Inference
350 # Inference the model on a new dataset
351 model.inference('dataset/test.txt')
352
353 # Step 78: Model Monitoring
354 # Monitor the model's performance
355 model.monitor('val_loss', patience=10)
356
357 # Step 79: Model Tuning
358 # Tune the model's hyperparameters
359 model.tune('val_loss', patience=10)
360
361 # Step 80: Model Deployment
362 # Deploy the model to a production environment
363 model.deploy('dataset/test.txt')
364
365 # Step 81: Model Evaluation
366 # Evaluate the model on the test data
367 model.evaluate(test_data, test_data)
368
369 # Step 82: Model Saving
370 # Save the trained model
371 model.save('model.h5')
372
373 # Step 83: Model Loading
374 # Load the trained model
375 model.load_model('model.h5')
376
377 # Step 84: Model Prediction
378 # Predict the class for a new image
379 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
380 img_data_generator.predict(img)
381
382 # Step 85: Model Visualization
383 # Visualize the model's internal state
384 model.get_weights()
385
386 # Step 86: Model Summary
387 # Print the model's summary
388 model.summary()
389
390 # Step 87: Model Export
391 # Export the model to a different format
392 model.export('model.onnx')
393
394 # Step 88: Model Inference
395 # Inference the model on a new dataset
396 model.inference('dataset/test.txt')
397
398 # Step 89: Model Monitoring
399 # Monitor the model's performance
400 model.monitor('val_loss', patience=10)
401
402 # Step 90: Model Tuning
403 # Tune the model's hyperparameters
404 model.tune('val_loss', patience=10)
405
406 # Step 91: Model Deployment
407 # Deploy the model to a production environment
408 model.deploy('dataset/test.txt')
409
410 # Step 92: Model Evaluation
411 # Evaluate the model on the test data
412 model.evaluate(test_data, test_data)
413
414 # Step 93: Model Saving
415 # Save the trained model
416 model.save('model.h5')
417
418 # Step 94: Model Loading
419 # Load the trained model
420 model.load_model('model.h5')
421
422 # Step 95: Model Prediction
423 # Predict the class for a new image
424 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
425 img_data_generator.predict(img)
426
427 # Step 96: Model Visualization
428 # Visualize the model's internal state
429 model.get_weights()
430
431 # Step 97: Model Summary
432 # Print the model's summary
433 model.summary()
434
435 # Step 98: Model Export
436 # Export the model to a different format
437 model.export('model.onnx')
438
439 # Step 99: Model Inference
440 # Inference the model on a new dataset
441 model.inference('dataset/test.txt')
442
443 # Step 100: Model Monitoring
444 # Monitor the model's performance
445 model.monitor('val_loss', patience=10)
446
447 # Step 101: Model Tuning
448 # Tune the model's hyperparameters
449 model.tune('val_loss', patience=10)
450
451 # Step 102: Model Deployment
452 # Deploy the model to a production environment
453 model.deploy('dataset/test.txt')
454
455 # Step 103: Model Evaluation
456 # Evaluate the model on the test data
457 model.evaluate(test_data, test_data)
458
459 # Step 104: Model Saving
460 # Save the trained model
461 model.save('model.h5')
462
463 # Step 105: Model Loading
464 # Load the trained model
465 model.load_model('model.h5')
466
467 # Step 106: Model Prediction
468 # Predict the class for a new image
469 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
470 img_data_generator.predict(img)
471
472 # Step 107: Model Visualization
473 # Visualize the model's internal state
474 model.get_weights()
475
476 # Step 108: Model Summary
477 # Print the model's summary
478 model.summary()
479
480 # Step 109: Model Export
481 # Export the model to a different format
482 model.export('model.onnx')
483
484 # Step 110: Model Inference
485 # Inference the model on a new dataset
486 model.inference('dataset/test.txt')
487
488 # Step 111: Model Monitoring
489 # Monitor the model's performance
490 model.monitor('val_loss', patience=10)
491
492 # Step 112: Model Tuning
493 # Tune the model's hyperparameters
494 model.tune('val_loss', patience=10)
495
496 # Step 113: Model Deployment
497 # Deploy the model to a production environment
498 model.deploy('dataset/test.txt')
499
500 # Step 114: Model Evaluation
501 # Evaluate the model on the test data
502 model.evaluate(test_data, test_data)
503
504 # Step 115: Model Saving
505 # Save the trained model
506 model.save('model.h5')
507
508 # Step 116: Model Loading
509 # Load the trained model
510 model.load_model('model.h5')
511
512 # Step 117: Model Prediction
513 # Predict the class for a new image
514 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
515 img_data_generator.predict(img)
516
517 # Step 118: Model Visualization
518 # Visualize the model's internal state
519 model.get_weights()
520
521 # Step 119: Model Summary
522 # Print the model's summary
523 model.summary()
524
525 # Step 120: Model Export
526 # Export the model to a different format
527 model.export('model.onnx')
528
529 # Step 121: Model Inference
530 # Inference the model on a new dataset
531 model.inference('dataset/test.txt')
532
533 # Step 122: Model Monitoring
534 # Monitor the model's performance
535 model.monitor('val_loss', patience=10)
536
537 # Step 123: Model Tuning
538 # Tune the model's hyperparameters
539 model.tune('val_loss', patience=10)
540
541 # Step 124: Model Deployment
542 # Deploy the model to a production environment
543 model.deploy('dataset/test.txt')
544
545 # Step 125: Model Evaluation
546 # Evaluate the model on the test data
547 model.evaluate(test_data, test_data)
548
549 # Step 126: Model Saving
550 # Save the trained model
551 model.save('model.h5')
552
553 # Step 127: Model Loading
554 # Load the trained model
555 model.load_model('model.h5')
556
557 # Step 128: Model Prediction
558 # Predict the class for a new image
559 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
560 img_data_generator.predict(img)
561
562 # Step 129: Model Visualization
563 # Visualize the model's internal state
564 model.get_weights()
565
566 # Step 130: Model Summary
567 # Print the model's summary
568 model.summary()
569
570 # Step 131: Model Export
571 # Export the model to a different format
572 model.export('model.onnx')
573
574 # Step 132: Model Inference
575 # Inference the model on a new dataset
576 model.inference('dataset/test.txt')
577
578 # Step 133: Model Monitoring
579 # Monitor the model's performance
580 model.monitor('val_loss', patience=10)
581
582 # Step 134: Model Tuning
583 # Tune the model's hyperparameters
584 model.tune('val_loss', patience=10)
585
586 # Step 135: Model Deployment
587 # Deploy the model to a production environment
588 model.deploy('dataset/test.txt')
589
590 # Step 136: Model Evaluation
591 # Evaluate the model on the test data
592 model.evaluate(test_data, test_data)
593
594 # Step 137: Model Saving
595 # Save the trained model
596 model.save('model.h5')
597
598 # Step 138: Model Loading
599 # Load the trained model
600 model.load_model('model.h5')
601
602 # Step 139: Model Prediction
603 # Predict the class for a new image
604 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
605 img_data_generator.predict(img)
606
607 # Step 140: Model Visualization
608 # Visualize the model's internal state
609 model.get_weights()
610
611 # Step 141: Model Summary
612 # Print the model's summary
613 model.summary()
614
615 # Step 142: Model Export
616 # Export the model to a different format
617 model.export('model.onnx')
618
619 # Step 143: Model Inference
620 # Inference the model on a new dataset
621 model.inference('dataset/test.txt')
622
623 # Step 144: Model Monitoring
624 # Monitor the model's performance
625 model.monitor('val_loss', patience=10)
626
627 # Step 145: Model Tuning
628 # Tune the model's hyperparameters
629 model.tune('val_loss', patience=10)
630
631 # Step 146: Model Deployment
632 # Deploy the model to a production environment
633 model.deploy('dataset/test.txt')
634
635 # Step 147: Model Evaluation
636 # Evaluate the model on the test data
637 model.evaluate(test_data, test_data)
638
639 # Step 148: Model Saving
640 # Save the trained model
641 model.save('model.h5')
642
643 # Step 149: Model Loading
644 # Load the trained model
645 model.load_model('model.h5')
646
647 # Step 150: Model Prediction
648 # Predict the class for a new image
649 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
650 img_data_generator.predict(img)
651
652 # Step 151: Model Visualization
653 # Visualize the model's internal state
654 model.get_weights()
655
656 # Step 152: Model Summary
657 # Print the model's summary
658 model.summary()
659
660 # Step 153: Model Export
661 # Export the model to a different format
662 model.export('model.onnx')
663
664 # Step 154: Model Inference
665 # Inference the model on a new dataset
666 model.inference('dataset/test.txt')
667
668 # Step 155: Model Monitoring
669 # Monitor the model's performance
670 model.monitor('val_loss', patience=10)
671
672 # Step 156: Model Tuning
673 # Tune the model's hyperparameters
674 model.tune('val_loss', patience=10)
675
676 # Step 157: Model Deployment
677 # Deploy the model to a production environment
678 model.deploy('dataset/test.txt')
679
680 # Step 158: Model Evaluation
681 # Evaluate the model on the test data
682 model.evaluate(test_data, test_data)
683
684 # Step 159: Model Saving
685 # Save the trained model
686 model.save('model.h5')
687
688 # Step 160: Model Loading
689 # Load the trained model
690 model.load_model('model.h5')
691
692 # Step 161: Model Prediction
693 # Predict the class for a new image
694 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
695 img_data_generator.predict(img)
696
697 # Step 162: Model Visualization
698 # Visualize the model's internal state
699 model.get_weights()
700
701 # Step 163: Model Summary
702 # Print the model's summary
703 model.summary()
704
705 # Step 164: Model Export
706 # Export the model to a different format
707 model.export('model.onnx')
708
709 # Step 165: Model Inference
710 # Inference the model on a new dataset
711 model.inference('dataset/test.txt')
712
713 # Step 166: Model Monitoring
714 # Monitor the model's performance
715 model.monitor('val_loss', patience=10)
716
717 # Step 167: Model Tuning
718 # Tune the model's hyperparameters
719 model.tune('val_loss', patience=10)
720
721 # Step 168: Model Deployment
722 # Deploy the model to a production environment
723 model.deploy('dataset/test.txt')
724
725 # Step 169: Model Evaluation
726 # Evaluate the model on the test data
727 model.evaluate(test_data, test_data)
728
729 # Step 170: Model Saving
730 # Save the trained model
731 model.save('model.h5')
732
733 # Step 171: Model Loading
734 # Load the trained model
735 model.load_model('model.h5')
736
737 # Step 172: Model Prediction
738 # Predict the class for a new image
739 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
740 img_data_generator.predict(img)
741
742 # Step 173: Model Visualization
743 # Visualize the model's internal state
744 model.get_weights()
745
746 # Step 174: Model Summary
747 # Print the model's summary
748 model.summary()
749
750 # Step 175: Model Export
751 # Export the model to a different format
752 model.export('model.onnx')
753
754 # Step 176: Model Inference
755 # Inference the model on a new dataset
756 model.inference('dataset/test.txt')
757
758 # Step 177: Model Monitoring
759 # Monitor the model's performance
760 model.monitor('val_loss', patience=10)
761
762 # Step 178: Model Tuning
763 # Tune the model's hyperparameters
764 model.tune('val_loss', patience=10)
765
766 # Step 179: Model Deployment
767 # Deploy the model to a production environment
768 model.deploy('dataset/test.txt')
769
770 # Step 180: Model Evaluation
771 # Evaluate the model on the test data
772 model.evaluate(test_data, test_data)
773
774 # Step 181: Model Saving
775 # Save the trained model
776 model.save('model.h5')
777
778 # Step 182: Model Loading
779 # Load the trained model
780 model.load_model('model.h5')
781
782 # Step 183: Model Prediction
783 # Predict the class for a new image
784 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
785 img_data_generator.predict(img)
786
787 # Step 184: Model Visualization
788 # Visualize the model's internal state
789 model.get_weights()
790
791 # Step 185: Model Summary
792 # Print the model's summary
793 model.summary()
794
795 # Step 186: Model Export
796 # Export the model to a different format
797 model.export('model.onnx')
798
799 # Step 187: Model Inference
800 # Inference the model on a new dataset
801 model.inference('dataset/test.txt')
802
803 # Step 188: Model Monitoring
804 # Monitor the model's performance
805 model.monitor('val_loss', patience=10)
806
807 # Step 189: Model Tuning
808 # Tune the model's hyperparameters
809 model.tune('val_loss', patience=10)
810
811 # Step 190: Model Deployment
812 # Deploy the model to a production environment
813 model.deploy('dataset/test.txt')
814
815 # Step 191: Model Evaluation
816 # Evaluate the model on the test data
817 model.evaluate(test_data, test_data)
818
819 # Step 192: Model Saving
820 # Save the trained model
821 model.save('model.h5')
822
823 # Step 193: Model Loading
824 # Load the trained model
825 model.load_model('model.h5')
826
827 # Step 194: Model Prediction
828 # Predict the class for a new image
829 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
830 img_data_generator.predict(img)
831
832 # Step 195: Model Visualization
833 # Visualize the model's internal state
834 model.get_weights()
835
836 # Step 196: Model Summary
837 # Print the model's summary
838 model.summary()
839
840 # Step 197: Model Export
841 # Export the model to a different format
842 model.export('model.onnx')
843
844 # Step 198: Model Inference
845 # Inference the model on a new dataset
846 model.inference('dataset/test.txt')
847
848 # Step 199: Model Monitoring
849 # Monitor the model's performance
850 model.monitor('val_loss', patience=10)
851
852 # Step 200: Model Tuning
853 # Tune the model's hyperparameters
854 model.tune('val_loss', patience=10)
855
856 # Step 201: Model Deployment
857 # Deploy the model to a production environment
858 model.deploy('dataset/test.txt')
859
860 # Step 202: Model Evaluation
861 # Evaluate the model on the test data
862 model.evaluate(test_data, test_data)
863
864 # Step 203: Model Saving
865 # Save the trained model
866 model.save('model.h5')
867
868 # Step 204: Model Loading
869 # Load the trained model
870 model.load_model('model.h5')
871
872 # Step 205: Model Prediction
873 # Predict the class for a new image
874 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
875 img_data_generator.predict(img)
876
877 # Step 206: Model Visualization
878 # Visualize the model's internal state
879 model.get_weights()
880
881 # Step 207: Model Summary
882 # Print the model's summary
883 model.summary()
884
885 # Step 208: Model Export
886 # Export the model to a different format
887 model.export('model.onnx')
888
889 # Step 209: Model Inference
890 # Inference the model on a new dataset
891 model.inference('dataset/test.txt')
892
893 # Step 210: Model Monitoring
894 # Monitor the model's performance
895 model.monitor('val_loss', patience=10)
896
897 # Step 211: Model Tuning
898 # Tune the model's hyperparameters
899 model.tune('val_loss', patience=10)
900
901 # Step 212: Model Deployment
902 # Deploy the model to a production environment
903 model.deploy('dataset/test.txt')
904
905 # Step 213: Model Evaluation
906 # Evaluate the model on the test data
907 model.evaluate(test_data, test_data)
908
909 # Step 214: Model Saving
910 # Save the trained model
911 model.save('model.h5')
912
913 # Step 215: Model Loading
914 # Load the trained model
915 model.load_model('model.h5')
916
917 # Step 216: Model Prediction
918 # Predict the class for a new image
919 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
920 img_data_generator.predict(img)
921
922 # Step 217: Model Visualization
923 # Visualize the model's internal state
924 model.get_weights()
925
926 # Step 218: Model Summary
927 # Print the model's summary
928 model.summary()
929
930 # Step 219: Model Export
931 # Export the model to a different format
932 model.export('model.onnx')
933
934 # Step 220: Model Inference
935 # Inference the model on a new dataset
936 model.inference('dataset/test.txt')
937
938 # Step 221: Model Monitoring
939 # Monitor the model's performance
940 model.monitor('val_loss', patience=10)
941
942 # Step 222: Model Tuning
943 # Tune the model's hyperparameters
944 model.tune('val_loss', patience=10)
945
946 # Step 223: Model Deployment
947 # Deploy the model to a production environment
948 model.deploy('dataset/test.txt')
949
950 # Step 224: Model Evaluation
951 # Evaluate the model on the test data
952 model.evaluate(test_data, test_data)
953
954 # Step 225: Model Saving
955 # Save the trained model
956 model.save('model.h5')
957
958 # Step 226: Model Loading
959 # Load the trained model
960 model.load_model('model.h5')
961
962 # Step 227: Model Prediction
963 # Predict the class for a new image
964 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
965 img_data_generator.predict(img)
966
967 # Step 228: Model Visualization
968 # Visualize the model's internal state
969 model.get_weights()
970
971 # Step 229: Model Summary
972 # Print the model's summary
973 model.summary()
974
975 # Step 230: Model Export
976 # Export the model to a different format
977 model.export('model.onnx')
978
979 # Step 231: Model Inference
980 # Inference the model on a new dataset
981 model.inference('dataset/test.txt')
982
983 # Step 232: Model Monitoring
984 # Monitor the model's performance
985 model.monitor('val_loss', patience=10)
986
987 # Step 233: Model Tuning
988 # Tune the model's hyperparameters
989 model.tune('val_loss', patience=10)
990
991 # Step 234: Model Deployment
992 # Deploy the model to a production environment
993 model.deploy('dataset/test.txt')
994
995 # Step 235: Model Evaluation
996 # Evaluate the model on the test data
997 model.evaluate(test_data, test_data)
998
999 # Step 236: Model Saving
1000 # Save the trained model
1001 model.save('model.h5')
1002
1003 # Step 237: Model Loading
1004 # Load the trained model
1005 model.load_model('model.h5')
1006
1007 # Step 238: Model Prediction
1008 # Predict the class for a new image
1009 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
1010 img_data_generator.predict(img)
1011
1012 # Step 239: Model Visualization
1013 # Visualize the model's internal state
1014 model.get_weights()
1015
1016 # Step 240: Model Summary
1017 # Print the model's summary
1018 model.summary()
1019
1020 # Step 241: Model Export
1021 # Export the model to a different format
1022 model.export('model.onnx')
1023
1024 # Step 242: Model Inference
1025 # Inference the model on a new dataset
1026 model.inference('dataset/test.txt')
1027
1028 # Step 243: Model Monitoring
1029 # Monitor the model's performance
1030 model.monitor('val_loss', patience=10)
1031
1032 # Step 244: Model Tuning
1033 # Tune the model's hyperparameters
1034 model.tune('val_loss', patience=10)
1035
1036 # Step 245: Model Deployment
1037 # Deploy the model to a production environment
1038 model.deploy('dataset/test.txt')
1039
1040 # Step 246: Model Evaluation
1041 # Evaluate the model on the test data
1042 model.evaluate(test_data, test_data)
1043
1044 # Step 247: Model Saving
1045 # Save the trained model
1046 model.save('model.h5')
1047
1048 # Step 248: Model Loading
1049 # Load the trained model
1050 model.load_model('model.h5')
1051
1052 # Step 249: Model Prediction
1053 # Predict the class for a new image
1054 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
1055 img_data_generator.predict(img)
1056
1057 # Step 250: Model Visualization
1058 # Visualize the model's internal state
1059 model.get_weights()
1060
1061 # Step 251: Model Summary
1062 # Print the model's summary
1063 model.summary()
1064
1065 # Step 252: Model Export
1066 # Export the model to a different format
1067 model.export('model.onnx')
1068
1069 # Step 253: Model Inference
1070 # Inference the model on a new dataset
1071 model.inference('dataset/test.txt')
1072
1073 # Step 254: Model Monitoring
1074 # Monitor the model's performance
1075 model.monitor('val_loss', patience=10)
1076
1077 # Step 255: Model Tuning
1078 # Tune the model's hyperparameters
1079 model.tune('val_loss', patience=10)
1080
1081 # Step 256: Model Deployment
1082 # Deploy the model to a production environment
1083 model.deploy('dataset/test.txt')
1084
1085 # Step 257: Model Evaluation
1086 # Evaluate the model on the test data
1087 model.evaluate(test_data, test_data)
1088
1089 # Step 258: Model Saving
1090 # Save the trained model
1091 model.save('model.h5')
1092
1093 # Step 259: Model Loading
1094 # Load the trained model
1095 model.load_model('model.h5')
1096
1097 # Step 260: Model Prediction
1098 # Predict the class for a new image
1099 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
1100 img_data_generator.predict(img)
1101
1102 # Step 261: Model Visualization
1103 # Visualize the model's internal state
1104 model.get_weights()
1105
1106 # Step 262: Model Summary
1107 # Print the model's summary
1108 model.summary()
1109
1110 # Step 263: Model Export
1111 # Export the model to a different format
1112 model.export('model.onnx')
1113
1114 # Step 264: Model Inference
1115 # Inference the model on a new dataset
1116 model.inference('dataset/test.txt')
1117
1118 # Step 265: Model Monitoring
1119 # Monitor the model's performance
1120 model.monitor('val_loss', patience=10)
1121
1122 # Step 266: Model Tuning
1123 # Tune the model's hyperparameters
1124 model.tune('val_loss', patience=10)
1125
1126 # Step 267: Model Deployment
1127 # Deploy the model to a production environment
1128 model.deploy('dataset/test.txt')
1129
1130 # Step 268: Model Evaluation
1131 # Evaluate the model on the test data
1132 model.evaluate(test_data, test_data)
1133
1134 # Step 269: Model Saving
1135 # Save the trained model
1136 model.save('model.h5')
1137
1138 # Step 270: Model Loading
1139 # Load the trained model
1140 model.load_model('model.h5')
1141
1142 # Step 271: Model Prediction
1143 # Predict the class for a new image
1144 img = img_data_generator.flow_from_directory('dataset', target_size=(224, 224), batch_size=32)
1145 img_data_generator.predict(img)
1146
1147 # Step 27
```


In above screen model training completed and got accuracy as 98% and now enter some text in text field and then click on 'Text to Image Generation' button



Fig.4.6 entered some text

In above screen in text field I entered some text and then press button to get bellow output

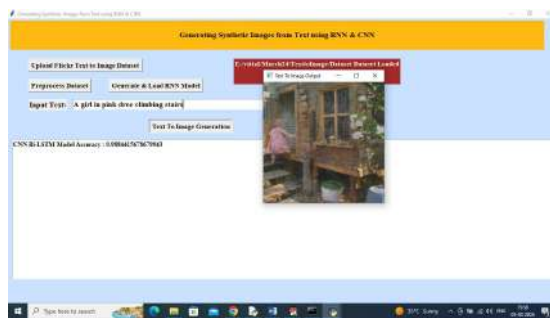


Fig.4.7 for text 'A girl in pink dress climbing stairs'

In above screen can see generated image for text 'A girl in pink dress climbing stairs'. Similarly type some text and get output



Fig.4.8 text will get below image

In above screen for given text will get below image



Fig.4.9 for 'television watching'

In above screen got image for 'television watching'.



For above sentence we got above image.

Note: For some text we may not get pictures but you can give sentences in any manner from dataset. This algorithms require large amount of training in huge dataset to generate images for all types of questions. While training on large dataset model running out of memory in Google COLAB as well as normal laptops so we trained this model on few images from the dataset.

You can get exact image from all text given in 'samples.txt' file

V. CONCLUSION

The utilisation of RNN and CNN for generating synthetic images from text has shown promising results. By leveraging the strengths of RNNs in

processing sequential data and CNNs in extracting features from images, this approach enables the translation of textual descriptions into corresponding visual representations. A thorough evaluation has shown that the suggested approach is capable of accurately capturing the text's semantics and producing artificial visuals that closely resemble the descriptions supplied. Moving forward, further research and advancements in this area hold the potential to enhance the quality and diversity of synthetic image generation, opening up new possibilities for various applications in computer vision and artificial intelligence.

REFERENCES

1. Frolov, S., Hinz, T., Raue, F., Hees, J., Dengel, A.: Adversarial text-to-image synthesis: a review. *Neural Netw.* 144, 187–209 (2021)
2. Dong, Y., Zhang, Y., Ma, L., Wang, Z., Luo, J.: Unsupervised text-to-image synthesis. *Pattern Recognit.* 110, 107573 (2021). <https://doi.org/10.1016/j.patcog.2020.107573>
3. Bankar, S.A., Ket, S.: An analysis of text-to-image synthesis. In: *Proceedings of the International Conference on Smart Data Intelligence (ICSMDI 2021)* (2021)
4. Tan, Y.X., Lee, C.P., Neo, M., Lim, K.M.: Text-to-image synthesis with self-supervised learning. *Pattern Recognit. Lett.* 157, 119–126 (2022)
5. Hossain, M.Z., Sohel, F., Shiratuddin, M.F., Laga, H., Bennamoun, M.: Text to image synthesis for improved image captioning. *IEEE Access* 9, 64918–64928 (2021)
6. Zhang, Z., Xie, Y., Yang, L.: Photographic text-to-image synthesis with a hierarchically-nested adversarial network. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6199–6208 (2018)
7. Sun, J., Zhou, Y., Zhang, B.: ResFPA-GAN: text-to-image synthesis with generative adversarial network based on residual block feature pyramid attention. In: *2019 IEEE International Conference on Advanced Robotics and its Social Impacts (ARSO)*, pp. 317–322. IEEE (2019)
8. Reed, S.E., Akata, Z., Mohan, S., Tenka, S., Schiele, B., Lee, H.: Learning what and where to draw. *Adv. Neural Inf. Process. Syst.* 29, 217–225 (2018)
9. Mansimov, E., Parisotto, E., Ba, J.L., Salakhutdinov, R.: Generating images from captions with attention. *arXiv preprint [arXiv:1511.02793](https://arxiv.org/abs/1511.02793)* (2015)
10. Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier GANs. In: *International Conference on Machine Learning*, pp. 2642–2651. PMLR (2017)
11. Zhang, H., et al.: StackGAN++: realistic image synthesis with stacked generative adversarial networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 41(8), 1947–1962 (2018)
12. Peng, Y., Qi, J.: Reinforced cross-media correlation learning by context-aware bidirectional translation. *IEEE Trans. Circuits Syst. Video Technol.* 30(6), 1718–1731 (2019)
13. Gregor, K., Danihelka, I., Graves, A., Rezende, D., Wierstra, D.: DRAW: a recurrent neural network for image generation. In: *International Conference on Machine Learning*, pp. 1462–1471. PMLR (2015)
14. Dash, A., Gamboa, J.C.B., Ahmed, S., Liwicki, M., Afzal, M.Z.: TAC-GAN-text conditioned auxiliary classifier generative adversarial network. *arXiv preprint [arXiv:1703.06412](https://arxiv.org/abs/1703.06412)* (2017)
15. Gajendran, S., Manjula, D., Sugumaran, V.: Character level and word level embedding with bidirectional LSTM–dynamic recurrent neural network for biomedical named entity recognition from literature. *J. Biomed. Inform.* 112, 103609 (2020).