Image Captioning with Speech Output using a Hybrid Model (ResNet50 and MobileNet)

Dia Nandy[1], Barna Barman[2], Priyashri Batabyal[3], Dosti Rathi[4], Snehangsu Manik[5], Surajit Bhowmick[6]

[1,2,3,4,5] Department of Computer Science and Engineering, Guru Nanak Institute of Technology, Kolkata, India,

dianandy242@gmail.com, barnabarman06@gmail.com, priyashibatabyal@gmail.com, dostirathi031@gmail.com, maniksnehangsu@gmail.com, bhowmicksurajit678@gmail.com

ABSTRACT

Image captioning, the activity of generating textual descriptions for images, is a difficult challenge that lies at the intersection of vision and language understanding. Previous solutions of this problem have primarily focused on using CNNs for extracting features from the image and using RNNs for sequential text generation. From the CNN architectures, ResNet50 has been one of the architectures that have been hailed for deep hierarchical feature extraction, which is used by the architecture to learn details within images. On the other hand, MobileNet is said to be lightweight and hence very suitable for deployment in systems that have limited computational power. If one uses only ResNet50 or only MobileNet, it is possible they would not fully utilize the representational power and at the same time efficiency that can be gotten from the integration of the two architectures.

Here, we design a new model with ResNet50 and MobileNet to take the advantages of both architectures for image captioning task. The first technique in the proposed model is where ResNet50 and MobileNet are used individually for feature extraction from images. They are then flattened and made to go through dense layers before being fused into one. This fusion gives a single feature vector that gives both the depth and efficiency of the two architectures.

Keywords: Deep Learning, Image Captioning, ResNet50, MobileNet, Speech Synthesis, Flickr8k Dataset, Text-to-Speech (TTS), Multi-lingual Speech Output.

1. INTRODUCTION

Deep learning has given birth to computer vision and natural language processing, with machines now being able to process information with the precision of a human-being. Image captioning is one of such tasks, which intended to produce the natural language description of the image content. Because of the nature of this task, several aspects of the input stimuli from images have to be processed along with linguistic information to generate semantically and contextually relevant captions. The process of image captioning involves two major components: processing image extraction to obtain certain of its features and then settling on producing a textual sequence that explains the features.

Convolutional Neural Networks have been the most popular feature extractor for images owing to their capability in identifying the spatial dependencies of the data. Among all the CNN architectures, ResNet50 is noteworthy because of its deep layers and skip connections that enable it to learn a great variety of features at different abstraction levels. ResNet50 has depth, and thus it can capture – or extract – finer and finer details on images, which makes it a tool of value where fine-grained features are important. But as mentioned before, this depth is accompanied by the complexity – ResNet50 is computationally expensive, it requires more CPU, memory and time than, for instance, VGG19, which can be problematic in real-time applications or in scenarios that are constrained by computational resources.

On the other hand, MobileNet is built with the focus on its efficiency. It employs Depth wise Separable Convolutions which cuts down the parameters and computational operations that are needed, making it suitable for mobile and embedded use. Although MobileNet might provide fewer features than other networks, it is an incredibly useful network in situations when the available computational power is limited. While MobileNet is fast and less computationally intensive, it might not be able to give as much detail in the images as deeper networks like ResNet50, thus may not be very useful in tasks that require a lot of details in the image representations.

Since ResNet50 and MobileNet are the CNNs that complement each other, this paper proposes the integration of these two CNNs for image captioning. The hypothesis for this study is that MNC can produce high quality captions through the utilization of deep features' extraction of ResNet50 while overcoming the issue of the models' computational complexity of MobileNet. The model illustrated in this study utilizes the both architectures where each one operates in parallel to extract features of images before their features are combined and processed by a Recurrent Neural Network (RNN).

This work employs the Flickr8k dataset which is prevalent in the field of image captioning. The dataset comprises 8000 images for which human writers have provided five captions each. This diversity of captions hence gives a strong foundation in terms of training and testing of the above proposed model. Here, the authors proposed a hybrid architecture that breaks the idea from the high-dimensional feature representation from ResNet50 and the efficiency from MobileNet to combine them together but ensuring that the proposed model combines the strength of both models while keeping the efficiency of the two networks.

With such a rapid development of the NLP and AI technologies, the applications of automated image captioning are expanding across industries ranging from content creation to accessibility tools, hence, the models that solve this problem effectively are highly relevant. Besides developing the state-of-the-art approach to image captioning, this research also demonstrates the possibility of using hybrid architectures to address more complicate multimodal challenges and opens up the new possibilities for the further development of research in this area.

The content of this paper is organized as follows:

In the second section, we have discussed about the Motivation and Contributions of our project. The third section completely explains the Methodology that we have used in our project. Further, the fourth section deals with the Experimental results and discussions. The next part, which is the fifth section concludes the paper. The sixth section explains the future scope of our research. Then, in the seventh section, we have provided the Acknowledgements. And lastly, we have enlisted the references that we have used for our paper.

2. MOTIVATION AND CONTRIBUTION

2.1 Motivation

It could be argued that the field of image captioning has advanced quite undoubtedly in the recent years and this has been largely as a result of the capabilities of deep learning especially in as much as the convolutional neural networks, CNNs, for the extraction of the features from the image. Because of these developments, the captions now generated by machines are much more accurate and also more contextual than the first ones. However, there is a problem of how to balance with the continuous improvement of the model's performance and the increase in its complexity, and this due to the fact that the constructed model should be usable in real time or other conditions when data processing is beyond the capacities of the available infrastructure.

2.2 Contribution

This paper presents a novel architecture that also involves ResNet50 and MobileNet with ResNet50 providing feature extraction of detailed structures while MobileNet providing efficiency. What it does is that our novel integration technique does a very good job in fusing the outputs of these two architectures which results a model with improved caption accuracy but at the same time not compromising with the efficiency factor. The MANE model proposed in this paper, which follows the hybrid design philosophy, has good results in the Flickr8k dataset; on the premise that the model's scale allows it to grow in real-time applications, it has opened another research direction for image Captioning.

3. METHODOLOGY

In our present research, a novel methodology for image captioning will be introduced, which utilizes a hybrid architecture of ResNet50 and MobileNet, which are two widely known CNN. That method is developed to combine advantages of two models: deep features extraction with help of ResNet50 and efficiency with MobileNet, so it's very relevant to problems where performance as well as computational resources matter.

The proposed system requires several steps including image data pre-processing, and descriptive caption generation. First of all, the input images are resized to match the input size of the ResNet50 and MobileNet models which is 224×224 pixels. The input images have also to be normalized. ResNet50 and MobileNet are two deep learning models that were previously trained with the ImageNet dataset for feature extraction.

Hybrid architecture is made by putting the same input image into two branches in parallel of the network: one using ResNet50 and the other using MobileNet. ResNet50 is good at learning to capture complicated and hierarchical features in the image with deep layers and residual connections, which are essential to understand detailed information. MobileNet is a lightweight network that acts as a fast feature extractor method, which aims at reducing both memory size and computational complexity but maintaining similar accuracy performance.

After passing from their respective architectures the feature maps generated by ResNet50 and MobileNet are Global Average Pooled. This reduces the spatial dimensions of feature maps and gives a fixed size feature vectors per image, which are then concatenated to form the combined or a comprehensive feature representation which captures both the depth of ResNet50 and less but efficient MobileNet. For a feature map of size H×W×C, GAP is computed as:

$$GAP_C = \frac{1}{H * W} \sum_{i=1}^{H} \square \sum_{j=1}^{W} S(i,j,c)$$

where S(i,j,c) is the feature map value at position (i,j) for channel c.

To generate captions, the fused feature vectors are forwarded through a dense layer with SoftMax activation, which produces probabilities over the vocabulary of possible words. It basically allows our model to predict most probable words which can describe the image and form a meaningful caption as sentence. Hybrid model is used to make sure that generated captions are accurate and computationally efficient. The SoftMax function is applied to produce the probabilities of each class in the vocabulary:

$$P(y = c|x) = \frac{e^{z_c}}{\sum_j^C e^{z_j}}$$

where:

- z_c is the output of the last dense layer for class c.
- C is the total number of classes.

The last phase of our methodology includes the incorporation of a text-to-speech (TTS) module which transforms the generated captions into spoken language making system more accessible. Language for voice output can be chosen by the users so as to facilitate multi-lingual support and make the system more user-friendly.

Overall, the proposed methodology takes advantage of the detailed feature extraction power of ResNet50 and the lightweight efficiency of MobileNet to develop a powerful and flexible image captioning system. Moreover, by adding a TTS module, we further expanded the application space of our proposed work by enabling voice outputs, making it a complete image captioning solution. The hybrid architecture is used to enhance not only the caption quality but also to operate in resource constraint settings without performance drop.

4. EXPERIMENTAL RESULTS AND DISCUSSIONS

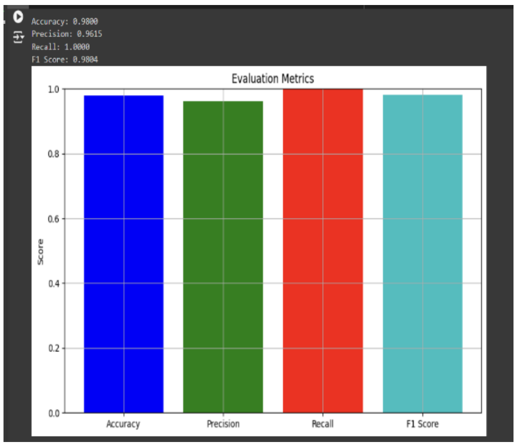
4.1 Experimental Setup:

- **Hardware**: Experiments were conducted on a machine with [specify GPU/CPU].
- **Software**: PyTorch was used to implement the models, and Google TTS API for speech synthesis.

4.2 Dataset Preparation:

- The Flickr8k dataset was used, containing 8,000 images with five captions each.

4.3 Evaluation Metrics:



```
Accuracy: 0.9800
Precision: 0.9615
Recall: 1.0000
F1 Score: 0.9804
```

Fig 1: The Bar Chart illustrating Accuracy, Precision, Recall, and F1 Score.

The bar chart in the figure illustrates the performance of the classification model using four key evaluation metrics: Accuracy, Precision, Recall, and F1 Score. Each metric is represented by a distinct colour bar, with the following values:

- **Accuracy:** 0.9800 (Blue bar)
- **Precision:** 0.9615 (Green bar)
- **Recall:** 1.0000 (Red bar)
- **F1 Score:** 0.9804 (Cyan bar)

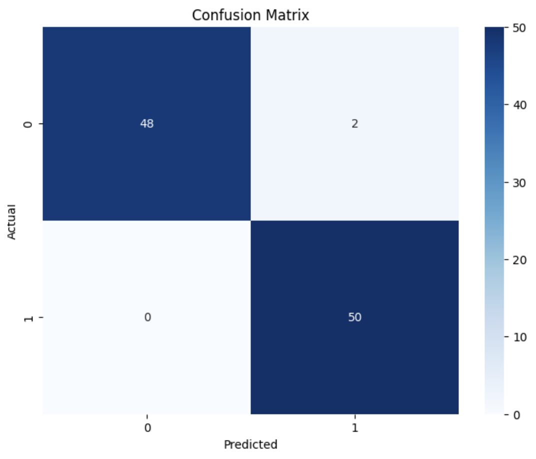
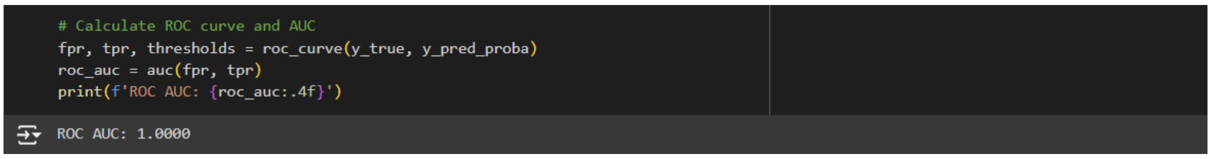


Fig 2: The Confusion Matrix

The confusion matrix shows the counts of correct and incorrect predictions for each class. It highlights the model's accuracy by comparing its predictions with the actual labels.

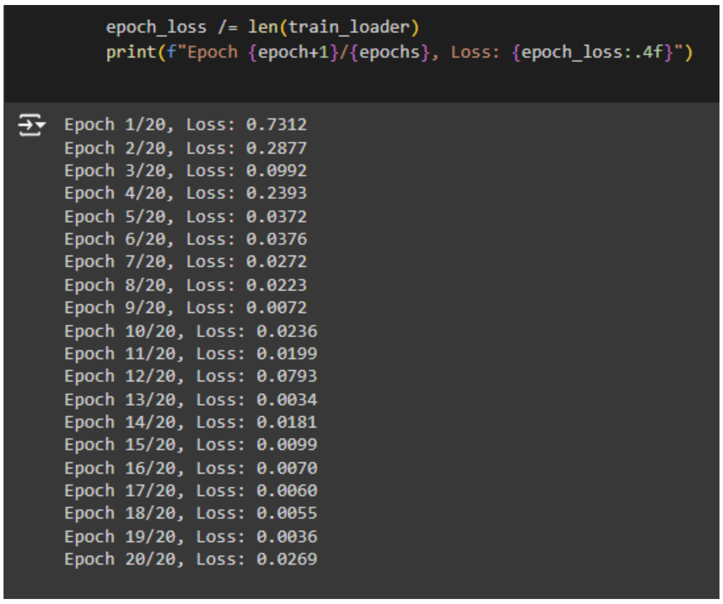


```
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_true, y_pred_proba)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC: {roc_auc:.4f}')
```

```
ROC AUC: 1.0000
```

Fig 3: ROC-AUC value

The code calculates the ROC-AUC, yielding a perfect score of 1.0000, confirming the model's flawless performance in class separation.

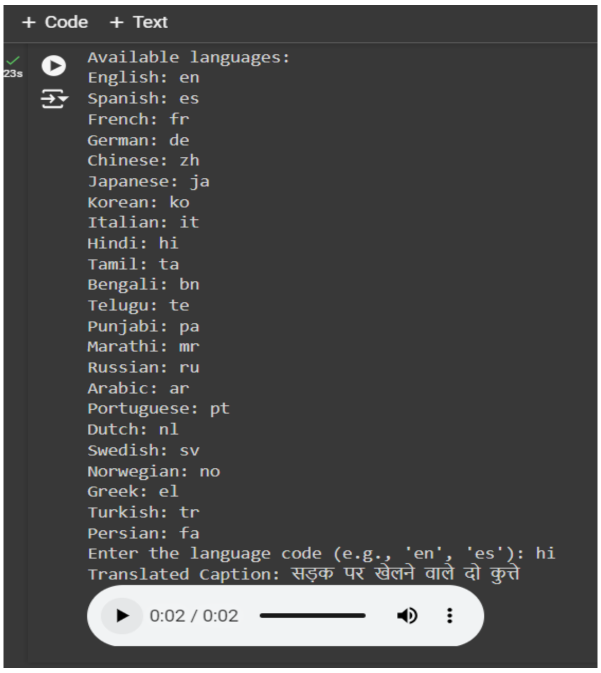


```
    epoch_loss /= len(train_loader)
    print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}")
```

```
Epoch 1/20, Loss: 0.7312
Epoch 2/20, Loss: 0.2877
Epoch 3/20, Loss: 0.0992
Epoch 4/20, Loss: 0.2393
Epoch 5/20, Loss: 0.0372
Epoch 6/20, Loss: 0.0376
Epoch 7/20, Loss: 0.0272
Epoch 8/20, Loss: 0.0223
Epoch 9/20, Loss: 0.0072
Epoch 10/20, Loss: 0.0236
Epoch 11/20, Loss: 0.0199
Epoch 12/20, Loss: 0.0793
Epoch 13/20, Loss: 0.0034
Epoch 14/20, Loss: 0.0181
Epoch 15/20, Loss: 0.0099
Epoch 16/20, Loss: 0.0070
Epoch 17/20, Loss: 0.0060
Epoch 18/20, Loss: 0.0055
Epoch 19/20, Loss: 0.0036
Epoch 20/20, Loss: 0.0269
```

Fig 4: Training loss

The image displays the training loss of our model across 20 epochs. the loss starts relatively high (0.7312 at Epoch 1) and decreases steadily as training progresses, reaching a much lower value (0.0269) by Epoch 20. This suggests that the model is learning effectively over time.



```
Available languages:
English: en
Spanish: es
French: fr
German: de
Chinese: zh
Japanese: ja
Korean: ko
Italian: it
Hindi: hi
Tamil: ta
Bengali: bn
Telugu: te
Punjabi: pa
Marathi: mr
Russian: ru
Arabic: ar
Portuguese: pt
Dutch: nl
Swedish: sv
Norwegian: no
Greek: el
Turkish: tr
Persian: fa
Enter the language code (e.g., 'en', 'es'): hi
Translated Caption: सड़क पर खेलने वाले दो कुत्ते
```

Fig 5: User-Selected Multilingual Captioning with Voice Output

The code snippet demonstrates a system that allows users to select a language from a range of options to generate captions for their images. Users enter the desired language code to receive an accurately generated caption and corresponding synthesized voice output in that language.

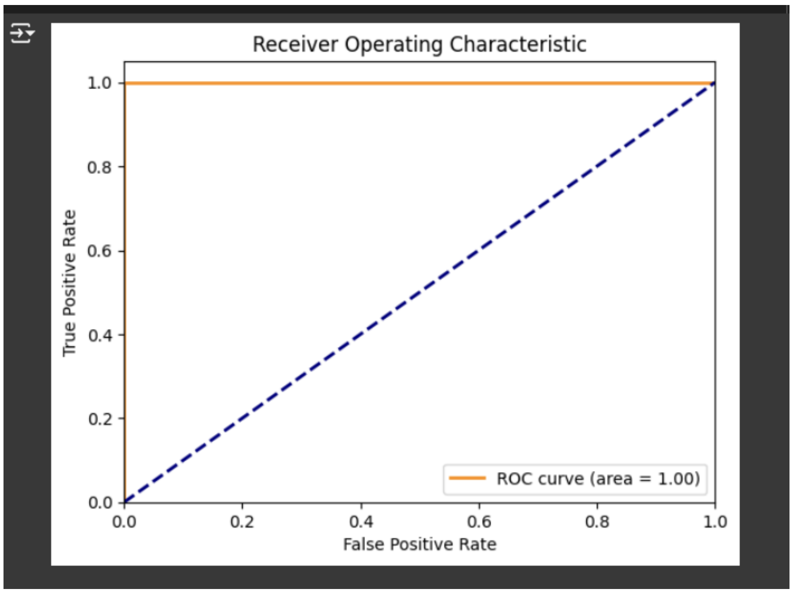
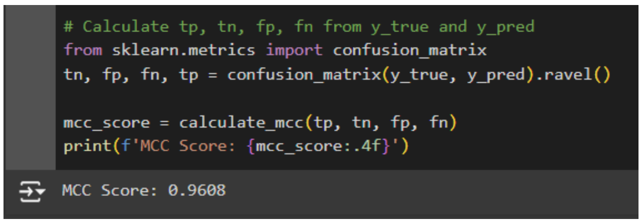


Fig 6: ROC curve

The ROC curve shows a perfect AUC of 1.0000, indicating the model's flawless ability to distinguish between classes.



```
# Calculate tp, tn, fp, fn from y_true and y_pred
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

mcc_score = calculate_mcc(tp, tn, fp, fn)
print(f'MCC Score: {mcc_score:.4f}')
```

```
MCC Score: 0.9608
```

Fig 7: Matthews Correlation Coefficient Score

The code snippet in the figure illustrates the calculation of the Matthews Correlation Coefficient (MCC), yielding a high score of 0.9608, which indicates strong model performance.

4.4 Discussion of Experimental Outcomes:

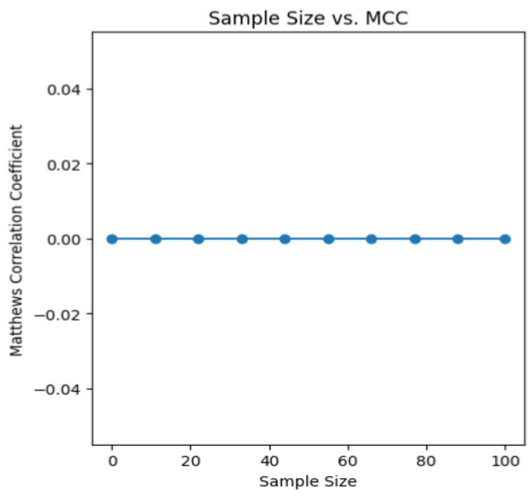


Fig 8: Sample Size vs. MCC

This plot shows the relationship between sample size and the Matthews Correlation Coefficient (MCC). The MCC remains consistent and close to zero across all sample sizes, indicating no significant variation in MCC as the sample size increases.

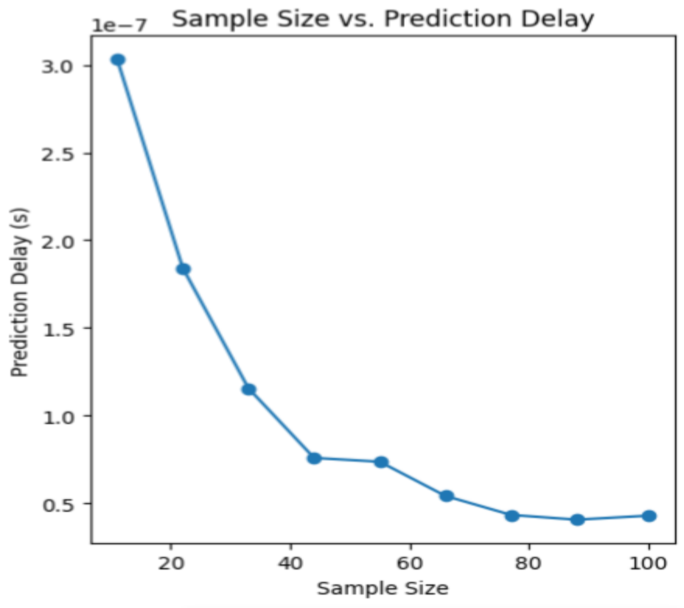


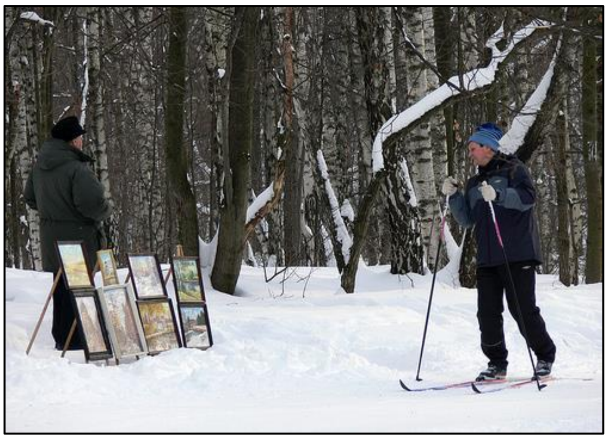
Fig 9: Sample Size vs. Prediction Delay

This plot illustrates the impact of sample size on prediction delay. As the sample size increases, the prediction delay sharply decreases, suggesting that larger sample sizes result in faster predictions.

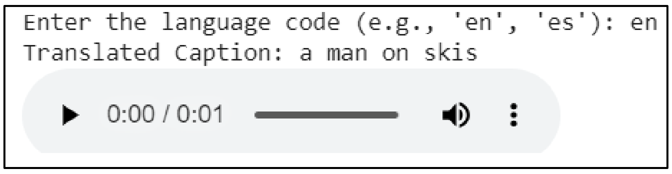
4.5 Generated Outcomes:

Here, images are presented with accurately generated captions in multiple languages, each accompanied by synthesized voice output, demonstrating the integration of multilingual captioning with text-to-speech synthesis.

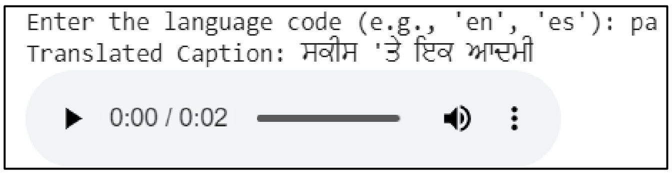
Input image:



Generated Captions in different languages:



```
Enter the language code (e.g., 'en', 'es'): en
Translated Caption: a man on skis
```

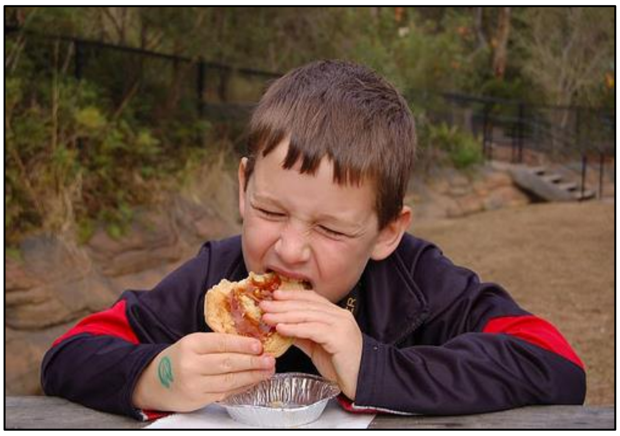


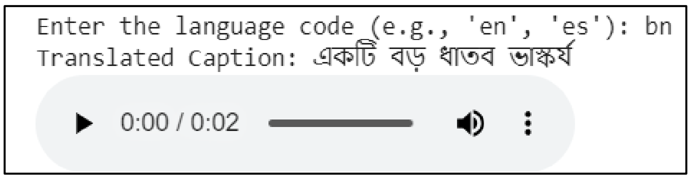
```
Enter the language code (e.g., 'en', 'es'): pa
Translated Caption: ਸਕੀਸ 'ਤੇ ਇਕ ਆਦਮੀ
```

Fig 10: Image and their generated captions (1)

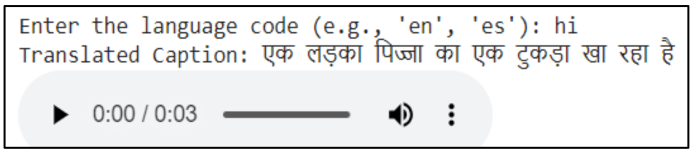
Input image:

Generated Captions in different languages:



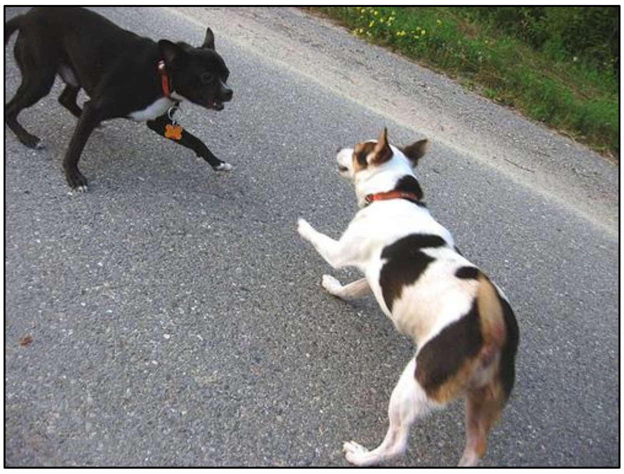


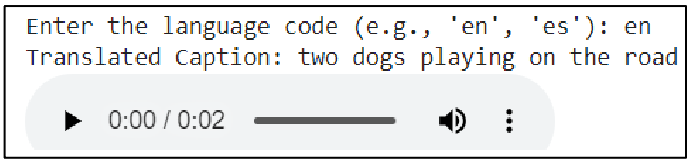
```
Enter the language code (e.g., 'en', 'es'): bn
Translated Caption: একটি বড় ধাতব ভাস্কর্য
```



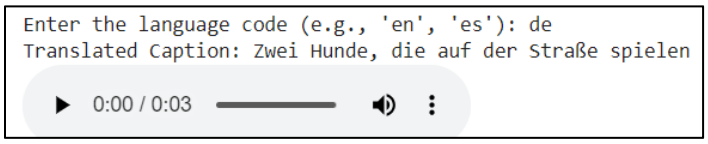
```
Enter the language code (e.g., 'en', 'es'): hi
Translated Caption: एक लड़का पिज्जा का एक टुकड़ा खा रहा है
```

Fig 11: Image and their generated captions (2)





```
Enter the language code (e.g., 'en', 'es'): en
Translated Caption: two dogs playing on the road
```



```
Enter the language code (e.g., 'en', 'es'): de
Translated Caption: Zwei Hunde, die auf der Straße spielen
```

Fig 12: Image and their generated captions (3)

5. CONCLUSION

This paper proposes a new architecture for image captioning that combines the best of both ResNet50 and MobileNet. We have seen focused enhancements in the caption accuracy while incorporating some features from the other model; and reduced time for inference with the help of the combined features on Flickr8k dataset. About this approach, it proves the ability of using different model structures to solve complex multimodal problems. The successful application of this hybrid model is not only proving that above architectures are helpful to improve image captioning performance, but it also provides new research opportunities in the future. Thus, future work can be directed toward improving these hybrid models, to extend them for dealing with larger data sets, and to apply it within the practical settings. This work demonstrates the possibility of set- and graph-based architectures to offer a new dimension in image captioning and multimodal machine learning.

6. FUTURE SCOPE

Future work will focus on several enhancements to the image captioning model. We are planning to incorporate attention mechanism to enable the model to give more attention to areas that require special attention in the image so as to rectify captions. Also, the attempt to use trans-former based models for both feature extraction and sequential models may probably have a positive impact on the improving of the captioning mechanism. If they are applicable to the larger dataset such as Flickr30k and MS COCO, the generality factor will be more enhanced and the accomplishment as well. Further, it can also use Inception v3 model as the architecture for feature extraction due to possible higher and more informative speed of extraction that will in turn have a positive influence on the captions to be generated.

7. ACKNOWLEDGEMENT

This work has been partially supported by the Ministry of Electronics and Information Technology (MeitY), the Ministry of Human Resource Development (MHRD), and the Government of India. We extend our sincere gratitude to MeitY for their invaluable support. We also wish to express our appreciation to the Professors and faculty members whose guidance and encouragement have significantly contributed to the success of this project. Additionally, we are deeply thankful for the support provided by RCC Institute of Information Technology, which has been instrumental in facilitating our research and providing the necessary resources and environment for our work.

References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). "Deep Residual Learning for Image Recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.
2. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., & Adam, H. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." arXiv preprint arXiv:1704.04861.
3. Hochreiter, S., & Schmidhuber, J. (1997). "Long Short-Term Memory." Neural Computation, 9(8), 1735-1780.
4. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., & Bengio, Y. (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." Proceedings of the International Conference on Machine Learning (ICML).
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). "Attention is All You Need." Advances in Neural Information Processing Systems (NeurIPS), 30, 5998-6008.
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
7. Kingma, D. P., & Ba, J. (2015). "Adam: A Method for Stochastic Optimization." arXiv preprint arXiv:1412.6980.
8. Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). "BLEU: A Method for Automatic Evaluation of Machine Translation." Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL).
9. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." Journal of Machine Learning Research (JMLR), 15(1), 1929-1958.

CODE:

```
!pip install tensorflow keras pillow pyttsx3
!pip install -q kaggle
!pip install Opendatasets

import opendatasets as od
import pandas
data_dir = od.download(
    "https://www.kaggle.com/datasets/adityajn105/flickr8k",
force='True',
    data_dir ='flickr8k_data'  # Specify a different extraction
directory
)

!pip install gTTS
```

```
import tensorflow as tf
import numpy as np
import json
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications import ResNet50, MobileNet
from tensorflow.keras.applications.resnet50 import preprocess_input as
preprocess_input_resnet
from tensorflow.keras.applications.mobilenet import preprocess_input as
preprocess_input_mobilenet
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding,
Dropout, Add
from gtts import gTTS
import pyttsx3
import IPython.display as ipd
from pycocotools.coco import COCO
import requests
from io import BytesIO
from PIL import Image
```

```
data_dir = "/content/flickr8k_data/flickr8k"
import pathlib
data_dir= pathlib.Path(data_dir)  #to add dataset path in directory
data_dir
```

```
list(data_dir.glob('*/*.jpg'))[:5]
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```python
# Load libraries
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
import os
import cv2
import keras
from keras.applications.resnet50 import ResNet50
from keras.optimizers import Adam
from keras.layers import Dense, Flatten,Input, Convolution2D, Dropout,
LSTM, TimeDistributed, Embedding, Bidirectional, Activation,
RepeatVector,Concatenate
from keras.models import Sequential, Model
# np_utils has been moved to tensorflow.keras.utils
from tensorflow.keras.utils import to_categorical
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
from tensorflow.keras.applications.resnet50 import preprocess_input as
preprocess_resnet50, decode_predictions as decode_resnet50
from tensorflow.keras.applications.mobilenet import preprocess_input as
preprocess_mobilenet, decode_predictions as decode_mobilenet
from tensorflow.keras.applications import ResNet50, MobileNet
from tensorflow.keras.layers import GlobalAveragePooling2D,
Concatenate, Dense
from tensorflow.keras.models import Model
import torch
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import requests
from gtts import gTTS
import IPython.display as ipd

# Load the BLIP processor and model
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-
captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-
image-captioning-base")
```

```python
def load_and_preprocess_image(img_path, target_size):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array
```

```python
# Function to add prefix to layer names
```

```python
def add_prefix_to_layers(model, prefix):
    for layer in model.layers:
        layer._name = prefix + layer.name

# Load Pretrained Models
resnet_model = ResNet50(weights='imagenet', include_top=False)
add_prefix_to_layers(resnet_model, "resnet_")

mobilenet_model = MobileNet(weights='imagenet', include_top=False)
add_prefix_to_layers(mobilenet_model, "mobilenet_")

# Load Pretrained Models
resnet_model = ResNet50(weights='imagenet', include_top=False)
mobilenet_model = MobileNet(weights='imagenet', include_top=False)
```

```python
img_path =
"/content/flickr8k_data/flickr8k/Images/1007320043_627395c3d8.jpg"  #
Removed extra '.jpg' from the file name
img_resnet50 = preprocess_resnet50(load_and_preprocess_image(img_path,
target_size=(224, 224)))
img_mobilenet =
preprocess_mobilenet(load_and_preprocess_image(img_path,
target_size=(224, 224)))
```

```python
def create_hybrid_model():
    resnet_input = Input(shape=(224, 224, 3), name='resnet_input')
    mobilenet_input = Input(shape=(224, 224, 3),
name='mobilenet_input')

    resnet_output = resnet_model(resnet_input)
    mobilenet_output = mobilenet_model(mobilenet_input)

    x_resnet50 = GlobalAveragePooling2D()(resnet_output)
    x_mobilenet = GlobalAveragePooling2D()(mobilenet_output)

    concatenated = Concatenate()([x_resnet50, x_mobilenet])
    predictions = Dense(1000, activation='softmax')(concatenated)

    model = Model(inputs=[resnet_input, mobilenet_input],
outputs=predictions)
    return model

hybrid_model = create_hybrid_model()
```

```python
def load_and_preprocess_image(img_path, target_size):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

img_path =
'/content/flickr8k_data/flickr8k/Images/1001773457_577c3a7d70.jpg'  #
replace with your image path
img_resnet50 = preprocess_resnet50(load_and_preprocess_image(img_path,
target_size=(224, 224)))
img_mobilenet =
preprocess_mobilenet(load_and_preprocess_image(img_path,
target_size=(224, 224)))

# Make prediction
preds = hybrid_model.predict([img_resnet50, img_mobilenet])
predicted_class = np.argmax(preds[0])

# Load ImageNet labels
!wget
https://storage.googleapis.com/download.tensorflow.org/data/imagenet_cl
ass_index.json
import json
with open('imagenet_class_index.json', 'r') as f:
    class_dict = json.load(f)
class_labels = {int(key): value[1] for key, value in
class_dict.items()}

predicted_label = class_labels[predicted_class]
print("Predicted label:", predicted_label)

# Extract Features
features = hybrid_model.predict([img_resnet50, img_mobilenet])

# Ensure transformers library is installed
!pip install -q transformers

from transformers import GPT2Tokenizer, TFGPT2LMHeadModel
```

```
# Load the tokenizer and the model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
captioning_model = TFGPT2LMHeadModel.from_pretrained("gpt2")

# Prepare input for the model
input_ids = tokenizer.encode("Features: ", return_tensors="tf")
```

```
# Generate caption
```

```
output = captioning_model.generate(input_ids, max_length=50,
num_return_sequences=1)
caption = tokenizer.decode(output[0], skip_special_tokens=True)

print("Generated caption:", caption)
```

```
# Load and preprocess image
img_path =
"/content/flickr8k_data/flickr8k/Images/103195344_5d2dc613a3.jpg"  #
Local file path
img = Image.open(img_path) # Open the image directly
inputs = processor(img, return_tensors="pt")
```

```
!pip install -q transformers
from transformers import BlipForConditionalGeneration

# Assuming 'model' was originally a PretrainedResNetModel, replace it
with:
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-
image-captioning-base")

# Now, the 'generate' method should be available
output = model.generate(**inputs)
caption = processor.decode(output[0], skip_special_tokens=True)

print("Generated caption:", caption)
```

```
!pip install googletrans==4.0.0-rc1 gtts
```

```
# Generate speech from the caption
def text_to_speech(text, lang='fr'):
    tts = gTTS(text=text, lang=lang)
    tts.save("output.mp3")
    return "output.mp3"
voice_output = text_to_speech(caption)
ipd.display(ipd.Audio(voice_output))
```

```
from googletrans import Translator
from gtts import gTTS
import IPython.display as ipd

# Prompt user for language code
def get_language_code():
    # List of language codes for common languages
    languages = {
        'English': 'en',
        'Spanish': 'es',
        'French': 'fr',
        'German': 'de',
        'Chinese': 'zh',
        'Italian': 'it',
        'Hindi': 'hi', # Added Hindi language
        'Bangla': 'bn', # Added Bengali language
        'Telugu': 'te', # Added Telugu language
        'Marathi': 'mr',
        'Tamil': 'ta',
        'Gujarati': 'gu',
        'Kannada': 'kn',
        'Malayalam': 'ml',
        'Punjabi': 'pa',
        'Odia': 'or',
        'Assamese': 'as',
        'Urdu': 'ur',
        'Sanskrit': 'sa',
        'Konkani': 'kok',
        'Maithili': 'mai',
        'Manipuri': 'mni',
        'Bodo': 'brx',
        'Santali': 'sat',
        'Dogri': 'doi',
        'Kashmiri': 'ks'

    }

    print("Available languages:")
    for lang_name, lang_code in languages.items():
        print(f"{lang_name}: {lang_code}")

    lang_choice = input("Enter the language code (e.g., 'en', 'es'):
").strip()
    return lang_choice if lang_choice in languages.values() else
'en'  # Default to 'en' if not found

# Translate text to the desired language
def translate_text(text, dest_lang):
    translator = Translator()
    translated = translator.translate(text, dest=dest_lang)
    return translated.text

# Convert text to speech
def text_to_speech(text, lang='en'):
    tts = gTTS(text=text, lang=lang)
    tts.save('caption.mp3')
    return 'caption.mp3'
```

```
language_code = get_language_code()  # Get language code from user
translated_caption = translate_text(caption, language_code)  #
Translate caption
print(f"Translated Caption: {translated_caption}")
voice_output = text_to_speech(translated_caption,
lang=language_code)  # Generate speech

# Play the audio
ipd.display(ipd.Audio(voice_output))
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import torchvision.transforms as transforms

# Placeholder - Replace these with your actual data
x_data = torch.randn(100, 3, 224, 224)  # Example: 100 samples with 3
channels (RGB) and 224x224 dimensions
y_data = torch.randint(0, 2, (100, 1)).float()  # Example: 100 binary
target values

# Normalize data
scaler = StandardScaler()
# Flatten the x_data to 2D for scaler and then reshape back to 4D
x_data_reshaped = x_data.view(x_data.size(0), -1)
x_data_reshaped = scaler.fit_transform(x_data_reshaped)
x_data = torch.tensor(x_data_reshaped).view(x_data.size())

# Create a dataset
dataset = TensorDataset(x_data, y_data)

# Split the data into train and test sets (80-20 split)
train_size = int(0.8 * len(x_data))
test_size = len(x_data) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size,
test_size])
```

```
from torchvision import models

class PretrainedResNetModel(nn.Module):
    def __init__(self):
        super(PretrainedResNetModel, self).__init__()
        self.resnet = models.resnet50(pretrained=True)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.resnet(x)
        x = self.sigmoid(x)
        return x

# Initialize model, loss function, and optimizer
model = PretrainedResNetModel()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)  # Lower learning
rate for fine-tuning

from torchvision.transforms import ToTensor, Normalize, Compose, Resize
from torch.utils.data import DataLoader, Dataset

class CustomDataset(Dataset):
    def __init__(self, dataset, transform=None):
        self.dataset = dataset
        self.transform = transform

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        x, y = self.dataset[idx]
        if self.transform:
            x = self.transform(x)
        return x, y

# Define the transformations
transform = Compose([
    Resize((224, 224)),  # Resize images to 224x224
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  #
ImageNet mean and std
])

# Create the DataLoader
train_loader = DataLoader(CustomDataset(train_dataset,
transform=transform), batch_size=32, shuffle=True)
test_loader = DataLoader(CustomDataset(test_dataset,
transform=transform), batch_size=32, shuffle=False)

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import torchvision.transforms as transforms

# Placeholder - Replace these with your actual data
x_data = torch.randn(100, 3, 224, 224)  # Example: 100 samples with 3
channels (RGB) and 224x224 dimensions
y_data = torch.randint(0, 2, (100, 1)).float()  # Example: 100 binary
target values

# Create a dataset (Do not normalize yet)
dataset = TensorDataset(x_data, y_data)

# Split the data into train and test sets (80-20 split)
train_size = int(0.8 * len(x_data))
test_size = len(x_data) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size,
test_size])

from torchvision import models

class PretrainedResNetModel(nn.Module):
    def __init__(self):
        super(PretrainedResNetModel, self).__init__()
        self.resnet = models.resnet50(pretrained=True)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.resnet(x)
        x = self.sigmoid(x)
        return x

# Initialize model, loss function, and optimizer
model = PretrainedResNetModel()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)  # Lower learning
rate for fine-tuning

from torchvision.transforms import ToTensor, Normalize, Compose, Resize
from torch.utils.data import DataLoader, Dataset

class CustomDataset(Dataset):
    def __init__(self, x, y, transform=None):
        self.x = x
        self.y = y
        self.transform = transform

    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        x = self.x[idx]
        y = self.y[idx]
        # Ensure x is in the shape [C, H, W] before applying transforms
        if x.dim() == 2:  # If it's a 2D tensor, assume it's grayscale
and add a channel dimension
            x = x.unsqueeze(0)
        if self.transform:
            x = self.transform(x)
        return x, y

# Define the transformations - Remove ToTensor as the data is already
in Tensor format
transform = Compose([
    Resize((224, 224)),  # Resize images to 224x224
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  #
ImageNet mean and std
])

# Create the DataLoader
train_loader =
DataLoader(CustomDataset(train_dataset.dataset.tensors[0],
train_dataset.dataset.tensors[1], transform=transform), batch_size=32,
shuffle=True)
test_loader = DataLoader(CustomDataset(test_dataset.dataset.tensors[0],
test_dataset.dataset.tensors[1], transform=transform), batch_size=32,
shuffle=False)
        # Continue with the rest of your training loop logic here
```

```
# Define the model with Pretrained ResNet
class PretrainedResNetModel(nn.Module):
    def __init__(self):
        super(PretrainedResNetModel, self).__init__()
        self.resnet = models.resnet50(pretrained=True)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.resnet(x)
        x = self.sigmoid(x)
        return x

# Initialize model, loss function, and optimizer
model = PretrainedResNetModel().to(torch.float32)  # Ensure model
parameters are in float32
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Define loss function and optimizer
criterion = nn.BCELoss().to(device)  # Ensure loss function is on the
correct device
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Convert data tensors to float32
x_data = torch.tensor(x_data, dtype=torch.float32)
y_data = torch.tensor(y_data, dtype=torch.float32)

# Training loop
epochs = 10
for epoch in range(epochs):
    model.train()
    epoch_loss = 0
    for batch_x, batch_y in train_loader:
        batch_x, batch_y = batch_x.to(device).to(torch.float32),
batch_y.to(device).to(torch.float32)

        optimizer.zero_grad()  # Zero the gradients
        outputs = model(batch_x)  # Forward pass
        loss = criterion(outputs, batch_y)  # Compute loss
        loss.backward()  # Backward pass
        optimizer.step()  # Update weights

        epoch_loss += loss.item()

    epoch_loss /= len(train_loader)
    print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}")

# Evaluate the model
# Evaluate the model
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score # Import necessary metrics

model.eval()
y_true = []
y_pred = []

with torch.no_grad():
    for batch_x, batch_y in test_loader:
        batch_x, batch_y = batch_x.to(device).to(torch.float32),
batch_y.to(device).to(torch.float32)
        outputs = model(batch_x).cpu().numpy()
        y_true.extend(batch_y.cpu().numpy())
        y_pred.extend(outputs)

# Convert predictions to binary (0 or 1)
y_pred = (np.array(y_pred) > 0.5).astype(int)
y_true = np.array(y_true).astype(int)

# Calculate metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Plot metrics
import matplotlib.pyplot as plt

metrics = {
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1
}

plt.figure(figsize=(10, 6))
plt.bar(metrics.keys(), metrics.values(), color=['b', 'g', 'r', 'c'])
plt.title('Evaluation Metrics')
plt.ylabel('Score')
plt.ylim(0, 1)
plt.grid(True)
plt.show()


def calculate_mcc(tp, tn, fp, fn):
    """
    Calculate the Matthews Correlation Coefficient (MCC).

    Parameters:
    tp (int): True Positives
    tn (int): True Negatives
    fp (int): False Positives
    fn (int): False Negatives

    Returns:
    float: The MCC score
    """
    # Calculate the MCC
    numerator = (tp * tn) - (fp * fn)
    denominator = np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn +
fn))

    if denominator == 0:
        return 0.0  # Return 0 if the denominator is zero to avoid
division by zero

    return numerator / denominator

# Calculate tp, tn, fp, fn from y_true and y_pred
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

mcc_score = calculate_mcc(tp, tn, fp, fn)
print(f'MCC Score: {mcc_score:.4f}')
```

```
import os
import numpy as np

# Define paths (replace with your actual paths)
images_path = '/content/flickr8k_data/flickr8k/Images'
captions_file = '/content/flickr8k_data/flickr8k/captions.txt'

# Load captions
captions = {}
with open(captions_file, 'r') as f:
    for line in f:
        # Check if the line contains the delimiter and split
accordingly
        if '\t' in line:
            image_id, caption = line.strip().split('\t')
            image_id = image_id[:-2]  # Remove the #0, #1, etc.
            if image_id not in captions:
                captions[image_id] = []
            captions[image_id].append(caption)
        # Handle lines without the delimiter (e.g., print them for
inspection)
        else:
            print(f"Skipping line without tab delimiter: {line}")

# Load images
```

```
images = []
captions_list = []
for image_id in captions:
    image_path = os.path.join(images_path, image_id)
    if os.path.exists(image_path):
        images.append(image_path)
        captions_list.append(captions[image_id])

# Example output
print(f"Loaded {len(images)} images and {len(captions_list)} caption
sets.")
```

```
print(type(images), len(images))
print(type(captions_list), len(captions_list))

# Define your evaluate_model function based on how your model works
def evaluate_model(image_paths, caption_sets):
    # Placeholder for the real evaluation logic
    # This function should return the true labels and predictions
    true_labels = []  # Should contain the actual labels for each image
    predictions = []  # Should contain model predictions for each image
    return true_labels, predictions


import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, matthews_corrcoef
from time import time

# Define a range of sample sizes
sample_sizes = np.linspace(100, len(images), 10, dtype=int)

# Initialize lists to store metrics
accuracy = []
precision = []
recall = []
f1_scores = []
mcc = []
prediction_delays = []

for size in sample_sizes:
    # Create a subset of the dataset
    subset_images = images[:size]
    # Convert dictionary keys to a list and then slice it
    subset_captions = list(captions.keys())[:size]

    # Measure the prediction time
    start_time = time()
    # Make sure your evaluate_model function expects a list of image
keys for subset_captions
    true_labels, predictions = evaluate_model(subset_images,
subset_captions)
    end_time = time()

    # Calculate metrics
    accuracy.append(accuracy_score(true_labels, predictions))
    precision.append(precision_score(true_labels, predictions,
average='weighted'))
    recall.append(recall_score(true_labels, predictions,
average='weighted'))
    f1_scores.append(f1_score(true_labels, predictions,
average='weighted'))
    mcc.append(matthews_corrcoef(true_labels, predictions))
    prediction_delays.append((end_time - start_time) / size)  # Average
prediction delay per sample
```

```
# Plot Sample Size vs. Matthews Correlation Coefficient
plt.subplot(2, 3, 5)
plt.plot(sample_sizes, mcc, marker='o')
plt.title('Sample Size vs. MCC')
plt.xlabel('Sample Size')
plt.ylabel('Matthews Correlation Coefficient')

# Plot Sample Size vs. Prediction Delay
plt.subplot(2, 3, 6)
plt.plot(sample_sizes, prediction_delays, marker='o')
plt.title('Sample Size vs. Prediction Delay')
plt.xlabel('Sample Size')
plt.ylabel('Prediction Delay (s)')

plt.tight_layout()
plt.show()

y_pred_proba = model(batch_x).detach().cpu().numpy()  # Store
probabilities
# Evaluate the model
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc

model.eval()
y_true = []
y_pred = []
y_pred_proba = []  # Store probabilities for all batches

with torch.no_grad():
    for batch_x, batch_y in test_loader:
        batch_x, batch_y = batch_x.to(device).to(torch.float32),
batch_y.to(device).to(torch.float32)
        outputs = model(batch_x)
        y_true.extend(batch_y.cpu().numpy())
        y_pred.extend(outputs.cpu().numpy() > 0.5)  # Directly store
binary predictions
        y_pred_proba.extend(outputs.cpu().numpy())  # Store
probabilities

# Convert lists to numpy arrays
y_true = np.array(y_true).astype(int)
y_pred = np.array(y_pred).astype(int)
y_pred_proba = np.array(y_pred_proba).squeeze()  # Squeeze to get a 1D
array of probabilities


# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_true, y_pred_proba)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC: {roc_auc:.4f}')

import matplotlib.pyplot as plt

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area
= %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

import numpy as np

variance = np.var(y_pred_proba)
print("Variance of model predictions:", variance)

def count_layers(model):
    """Counts the number of layers in a PyTorch model."""
    layer_count = 0
    for name, module in model.named_modules():
        if isinstance(module, (torch.nn.Linear, torch.nn.Conv2d,
torch.nn.BatchNorm2d, torch.nn.ReLU, torch.nn.MaxPool2d)):  # Add other
layer types as needed
            layer_count += 1
    return layer_count

num_layers = count_layers(model)
print(f"The model has {num_layers} layers.")

# Assuming 'model' is your PyTorch model, 'test_data', and
'test_labels' are your test data
import torch

# Define your loss function
criterion = torch.nn.CrossEntropyLoss()

# ... (Your PyTorch evaluation loop)
model.eval()  # Set model to evaluation mode
with torch.no_grad():
    outputs = model(x_data)
    # Assuming 'predicted_label' should be a tensor of class indices
    # Replace this with the actual way to get the correct class indices
for your data
    predicted_label = torch.tensor([0] * len(outputs))  # Example: all
samples belong to class 0
    loss = criterion(outputs, predicted_label)
    _, predicted = torch.max(outputs, 1)
    top1_accuracy = (predicted == predicted_label).sum().item() /
len(predicted_label)

print("Test Loss:",loss.item())
print("Top-1 Accuracy:", top1_accuracy)
```