

Deep Learning Methods for Video to Text Converter Applications with Phytorch Library

Komang Ayu Triana Indah^a, Ida Bagus Putra Manuaba and I Komang Wiratama
Department of Electrical Engineering, Bali State Polytechnic, South Kuta, Badung, Bali, Indonesia

Keywords: Deep Learning, Video Converter, Phytorch Library.

Abstract: One of the core elements of the concept of computer vision consists of image classification, object detection and segmentation. The multi-task deep learning method is implemented in the process of converting images into text through a multi-layer process to express complex image understanding. In this study, the system will convert the video into text and sentences. This synchronization is based on the Multitask Deep Learning method that combines the Convolutional Neural Network (CNN) system in the image area, Recurrent Neural Network (RNN) with LSTM (Long Short Term Memory) in the sentence area, CCN (Caption Content Network) and RCN (Recurrent Convolutional Network) on the labeling process and the relationship between objects as well as with a structured goal that aligns the two modalities through multimodal embedding. PyTorch is an extension of the Torch Framework which was originally written in the Lua programming language. The syntax that PyTorch uses is not much different in terms of functionality compared to other frameworks. Testing the results of converting images into text based on Multi Task Deep Learning with the RNN method using LSTM or BERT with scoring using f1-score, precision and recall. The results will be plotted using AUC (Area Under The Curve) and ROC (Receivers Operating Characteristics)graphs.

1 INTRODUCTION

Computer vision is an automated process that integrates a large number of processes for visual perception, such as image acquisition, image processing, recognition, and decision making. Computer vision is expected to have a high level of ability as a human visual. These capabilities include:

- Detection of objects, determine objects at the scene and their boundaries
- Recognition, label the object.
- Description, assigns properties to the object.
- 3D Inference, interprets a 3D scene from a 2D view.
- Interpreting motion, interpreting motion.


Computer vision combines cameras, edge or cloud-based computing, software, and artificial intelligence (AI) so the system can “see” and identify objects. Intel has a comprehensive portfolio of AI deployment technologies, including CPUs for general-purpose processing, computer vision, and vision processing units (VPUs) for acceleration. Computer vision systems that are useful in a variety of environments can quickly identify objects and people, analyze

audience demographics, inspect production outputs, and many other things. (L. R. Jácome-Galarza, 2020).

2 DEEP LEARNING

Development of Deep Learning and Image Processing methods through a system capable of solving high-level image understanding problems to change visual images in the context of sentences

The development of machine learning technology with multi-task deep learning uses several methods including the Convolutional Neural Network (CNN) which functions to extract visual features in the form of images, through the Recurrent Neural Network (RNN), especially the LSTM type for captioning various image areas of object elements by detecting relationships between objects. object. (S. Bai et al, 2018).

^a <https://orcid.org/0000-0003-3496-4484>

2.1 Multi Task Learning

Multi Task Learning (MTL) is used in machine learning applications, ranging from natural language processing, speech recognition, computer vision, and drug discovery. MTL comes in many forms: study together, learn to learn, and learn by task are just a few of the names used to refer to it. Generally, after optimizing more than one function it effectively performs multi-task learning (as opposed to task learning. The most commonly used way to perform multi-task learning in multi-task neural networks is usually done by sharing hidden layer parameters either hard or soft. hard parameters is the most commonly used approach for MTL in neural networks.(O. Sener and V. Koltun, 2018).

2.2 Deep Relationship Network

In MTL for computer vision, the approach often shares convolution layers, while learning the task-specific full connected layers by leveraging these models with Deep Relationship Networks. In addition to the shared structure and special layers, which can be seen in Figure 1, they placed the previous matrix on a fully connected layer, which allowed the model to study the relationships between tasks, similar to some Bayesian models.

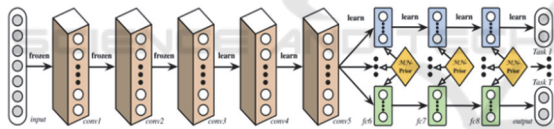


Figure 1: Deep Relationship Network with convolutional layers together and fully connected with prior matrices (X. Liu, P. He, W. Chen, 2019).

2.3 Recurrent Neural Network (RNN)

Generative image modeling is a major problem in unsupervised learning. Probabilistic density models can be used for a variety of tasks that range from image compression and reconstruction forms such as image inpainting (eg, see Figure 1) and deblurring, to the creation of new images. When the model is conditioned on external information, possible applications may also include creating images based on text descriptions or simulating future frames in planning tasks. One of the great advantages of generative modeling is that there is practically a large amount of image data available to study. However, because the images are high-dimensional and highly structured, estimating the natural image distribution

becomes very challenging. A glance at an image is enough for a human to show and explain a large amount of detail about a visual scene, however, this ability becomes an elusive task for visual recognition models. In this model the input process is a set of pictures and descriptions of sentences that match the following figure 2. (A. Karpathy and L. Fei-Fei, 2019).

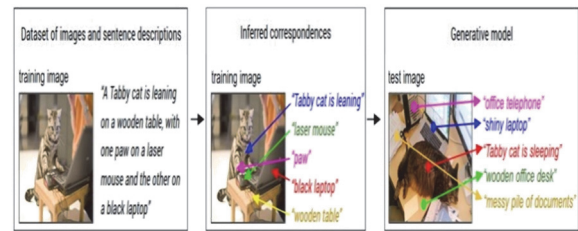


Figure 2: Model Set of Drawings and Appropriate Description (A. Karpathy and L. Fei-Fei, 2019).

One of the most important constraints in generative modeling is building the original occlusion solution model in Figure 2. Completion images are sampled from Pixel RNN. This exchange has produced a wide variety of generative models, each with its advantages. Most of the work focuses on stochastic latent variable models such as VAE which aim to extract meaningful representations, but are often accompanied by difficult inference steps that can hinder their performance. One effective approach to modeling the combined distribution of pixels in an image is to display them as the product of a conditional distribution. The resulting Pixel RNN consists of up to twelve two-dimensional layer Long Short Term Memory (LSTM) memories. The first type is the Row LSTM layer where convolution is applied along each row a similar technique is described in (A. Karpathy and L. Fei-Fei, 2017).

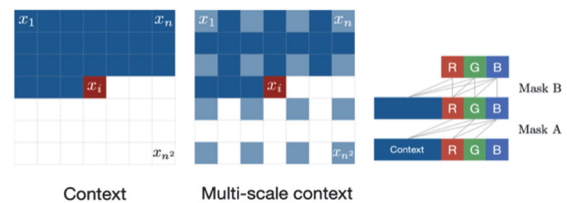


Figure 3: Model Set of Drawings and Appropriate Description.

Figure 3 can be explained as follows:

Left : To generate x_i pixels one condition on all previously created pixels to the left and above x_i .

Center : To generate pixels in the multi-scale case, we can also condition the sub-sampled image pixels (in light blue).

Right : Connectivity diagram inside a masked convolution. In the first layer, each RGB channel is connected to the previous channel and to the context, but is not connected to the channel itself. In the next layer, the channel is also connected to the subtitle.

3 SYSTEM AND ARCHITECTURAL MODEL

Author The architectural model of understanding imagery in the context of language can be divided into 3 processes:

1. Detect Objects and text areas.
2. The overall system architecture framework which is the relationship between object features, subject features, caption features and context features.
3. Object and network proposal caption region As represented in the following image simulation.

3.1 Objects and Text Areas

As shown in Figure 4 below, the initial process begins with video extraction by determining object detection and the detected text area on a scene graph. Then determine the Caption Region, Relationship Region, and Object Region. Furthermore, feature extraction consisting of Caption Feature, Relationship Feature and Object Feature and combining/processing Relationship Feature, Caption Feature and Relationship Context Feature with CCN (Caption Context Network) method to produce Caption Feature and Caption Context Feature in a Caption Generation. The final part is to combine and process the Subject Feature, Relationship Feature and Object Feature with the RCN (Relationship Context Network) method to produce Relationship Detection and then extract the feature object to produce Object Detection.(D. Shin and I. Kim,2018).

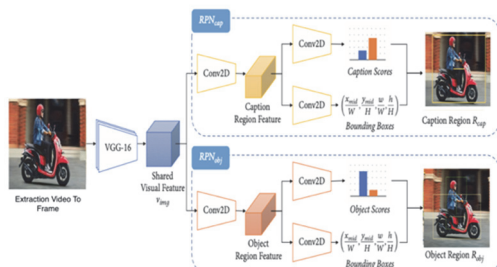


Figure 4: Object Detection Network and Text Region.

3.2 Dataset Exploration

Each The dataset used in this system uses the Tensorflow platform, with the Inception3 Architecture and the encoders and decodercnn methods. The Tensorflow dataset is a benchmark dataset for object detection of image segmentation. Detection of objects is done by regression (object restriction) and classification.(S. Ren, K. He, R. Girshick, 2017).

1. Setting up a Library / Data library (In research using Pytorch)
2. Downloading Data
3. Loading Data
4. Distribution of Objects in the Tensorflow Dataset
5. Utilities function (object constraint area)

3.3 Preprocessing

At this stage there are four processes, namely:

1. Image Representation
Sentence description can create a reference to the object and its attributes, so according to Girshick's method to detect objects in each image with RCNN. (L. R. Jácome-Galarza, 2020). CNNs were pre-trained on ImageNet and tuned to 200 classes by using the top detected locations in addition to the entire image and calculating the representation based on the Ib pixels within each bounding box as follows.

$$v = Wm[CNN\sqrt{c(Ib)}] + bm, (1)$$

Where CNN(Ib) converts the pixels inside the bounding box Ib into 4096 dimension activation of the fully connected layer just before the classifier. (S. Bai and S. An, 2018).

2. Sentence Representation
To establish the inter-modal relationship, then to represent the words in the sentence in the same h-dimensional embedding space as the image region. The simplest approach might be to project each shared word directly into this embedding. However, this approach does not consider word order and context information in sentences. To solve this problem, Bidirectional Recurrent Neural Network (BRNN) is used to calculate word representation. BRNN takes a sequence of N words (encoded in a 1-of-k representation) and transforms each into an h-dimensional vector. However, the representation of each word is enriched by the varying sized context around that word. (S. Aditya, Y. Yang, C. Baral, 2017). Using index $t = 1 \dots N$ to indicate the position

of a word in a sentence, the exact form of BRNN is the following equation:

$$x_t = W_w t(2)$$

$$e_t = f(W_e x_t + b_e) (3)$$

$$h_f = f(e_t + W_f h_f) (4)$$

$$h_t = f(e_t + W_b h_{t-1} + b_b) (5)$$

$$s_t = f(W_d(h_t + h_{t-1}) + b_d) (6)$$

Here, t is an indicator column vector that has one in the t -th word index in a vocabulary word. The W_w weight determines the word embedding matrix which we initialize with the word2vec weights of dimension 300 and is retained because of the overfitting problem. However, in practice we find little change in final performance when this vector is trained, even from random initialization. (A. Van Den Oord, N. Kalchbrenner, 2016).

3. Alignment Objective (Transforming Images and Sentences Into a Collection of Vectors in the Same h-Dimensional Space)

Since the control is at the picture and sentence level as a whole, the picture-sentence score is formulated as a function of the individual word-area scores. Intuitively, a picture-sentence pair must have a high match score if the words have convincing support in the picture. Karpathy et al's model (X. Liu, P. He, W. Chen, 2019) interprets the dot product $v_i^T s_t$ between the i -th region and the t -th word as a measure of similarity and uses it to determine the score between k image and l sentence as: (A. Karpathy and L. Fei-Fei, 2017).

4. This process is carried out by aligning the image from the training set and the corresponding sentence. We can interpret the quantity $v_i^T s_t$ as the non-normalized log probability of the t -th word describing the bounding box in the figure. However, in the end to create a text snippet rather than a single word, by aligning the extended and contiguous word order to a single bounding box. Note that the naive solution of assigning each word separately to the region with the highest score is insufficient because it causes the words to scatter inconsistently into different regions. The data encoding process is carried out with the ENDOCDERCNN (Encoding-Decoding CNN) Algorithm as shown in Figure 5 below.

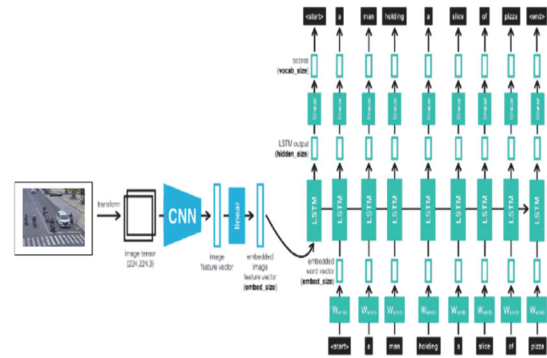


Figure 5: Data frame process flow with ENDOCDERCNN algorithm.

3.4 Mapping Method with Phytorch Library

The system is built on a web-based basis, which is implemented using the Python Programming Language and the PyTorch Library, as shown in Figure 9 below. PyTorch is a development of the Torch Framework which was originally written in the Lua programming language. The syntax that PyTorch uses is not too different from the functions in compared to other frameworks, PyTorch has a neater and simpler syntax. The following is an image of the mapping process using PyTorch to classify video data using CNN. PyTorch is a development of the Torch Framework which was originally written in the Lua programming language. The syntax that PyTorch uses is not too different from the functions in compared to other frameworks, PyTorch has a neater and simpler syntax. The following is an image of the mapping process using PyTorch to classify video data using CNN. The data has gone through pre-processing and feature extraction and is neatly stored in Numpy format with four dimensions measuring 1036 x 3 x 79 x 26. The ftrs.npy file has data features and the lbls.npy file has the appropriate data labels. There are 10 label classes with almost the same data for each class. (X. Liu, P. He, 2019).

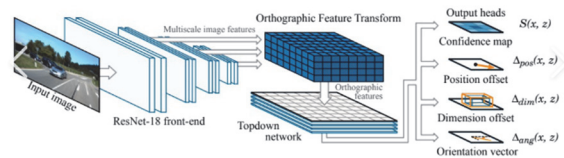


Figure 6: Basic Phytorch Workflow.

Modeling in PyTorch is not bound by any specific rules. The model used is usually in the form of a class with a forward(x) function to calculate the forward

propagation process. The torch.nn library is an important part that stores Neural Network functions. The model we use above is the CNN model with an additional 1 fully connected layer. It can be seen that we defined Conv2d, Maxpool2d and two linear layers. The forward function is used for the forward propagation process when data is inputted. It also appears that we use the ReLU activation function for each neuron.

4 PROCESS AND RESULT ANALYSIS

Python Programming Language and done on Google Collab platform. For an example of a 5 minute video, the first step is to import the library needed to understand the image, in this case using Tensor flow version 2. Next, download the dataset that has been stored on Google Drive. A short video dataset with a duration of 5 minutes as shown in Figure 10 below.



Figure 7: CCTV video recording data with a duration of 5 minutes.

The next step is to extract the previously downloaded dataset, then convert the mp4 video into an image using opencv. In the process of compiling the dataset, the dataset is created by describing each image in written form, and the text generation process is still in English according to the existing library. In the process of making the dataset, namely collecting images taken from CCTV recording videos, then a description of each object in the image is written and then stored in csv form to be loaded into the tensorflow dataset. Load the csv into python for inclusion in the tensorflow dataset then process the Train Caption. In Figure 10, the following is an example of an image dataset and its description.

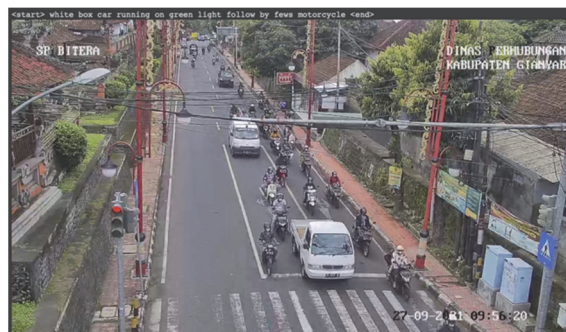


Figure 8: Image dataset and description.

Preprocess using InceptionV3 convert the image to the expected format of InceptionV3 by resizing the image to 299 x 299 after normalizing the image so that it is between the range -1 and 1. The next process is to make CNN Encoder and CNN Train. For details of the architecture above, we use a cnn encoder with a pre-trained model using resnet-50 to encode the image into an embedded image or feature vector. After that, the embedded image is entered into the RNN decoder. Texts also need to be pre-processed and prepared for training. In this example, to generate text, aim to build a model that predicts the next sentence token from the previous token, So I convert the text associated with any image into a tokenized word list, before sending it to a PyTorch tensor which we can use to train network. The next step is to divide the data into train and validate and create a tensorflow dataset using tf.data. These steps are performed by:

- Extract features from CNN InceptionV3 into vector shapes (8, 8, 2048).
- Converts to shape (64, 2048).
- Enter via CNN Encoder.
- Features in the decoder with RNN (here GRU) to generate images into writing.

After the process of sharing training data and validation data, the next step is the model training process. The two components of the model, namely the encoder and decoder, train these components together by passing the output of the encoder, which is a vector of latent space, to the decoder, which, in turn, is an iterative neural network. Example No. Of Epochs = 1 Batch Size = 32 and so on.

Based on the results of the training model by calculating losses from Epoch 1 to Epoch 100, we get Real Captions and Predicted Captions as shown in Figure 10 below.

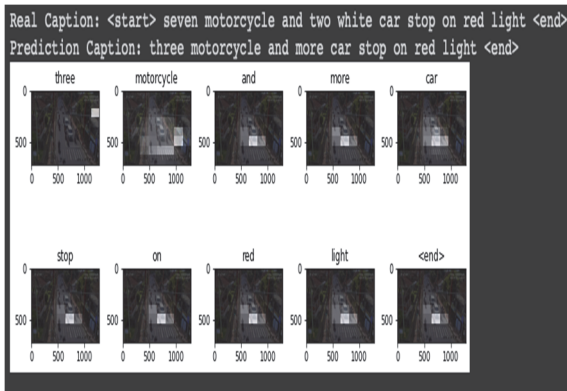


Figure 9: Real Caption and Prediction Caption.

5 MODEL TRAINING AND TESTING

Modeling in PyTorch is not bound by any specific rules. The model used is usually in the form of a class with a forward(x) function to calculate the forward propagation process. The torch.nn library is an important part that stores Neural Network functions. The model we use above is the CNN model with an additional 1 fully connected layer. It can be seen that we defined Conv2d, Maxpool2d and two linear layers. The forward function is used for the forward propagation process when data is inputted.

5.1 Split Dataset

We need to share the dataset that we have for training and testing purposes. Data sharing can be done using the help of the Scikit Learning library with a proportion of 80% for training and 20% for testing.

The text must be aligned to the left with the linespace set to single and in 9-point type. We need to share the dataset that we have for training and testing purposes. Data sharing can be done using the help of the Scikit Learning library with a proportion of 80% for training and 20% for testing.

5.2 Change to Tensor

The divided dataset is converted to a tensor and also starts initializing the initial parameters of the Model, such as optimization and evaluation algorithms. The criterion variable stores the evaluation function used, namely Cross Entropy. And the optimizer variable stores the optimization function that will be used, namely Adagrad. In the test using the epoch value of 1 to 100.

5.3 Testing Process

Testing is done inside torch.no_grad() to avoid accidentally calling autograd. When the program is run, an error value will be displayed during the training process. The error value will decrease indicating the training process is going well. After training, the accuracy value will be displayed. For data and epochs that have been defined, the author gets an accuracy of about 40-50%. In testing the training model, there is a comparison between the epoch and loss of each feature. Epoch testing on sample images is carried out from epochs 1 to 100, with loss results as shown in Figures 12 to 14 below.

Epoch 1 Batch 0 Loss 3.5881 1.7264183593545 Epoch 1 Loss 3.758418 Time taken for 1 epoch 38.16 sec	Epoch 11 Batch 0 Loss 2.3453 1.99994423194088 Epoch 11 Loss 1.999984 Time taken for 1 epoch 0.61 sec	Epoch 21 Batch 0 Loss 1.3943 1.503617181881836 Epoch 21 Loss 1.503617 Time taken for 1 epoch 0.37 sec	Epoch 31 Batch 0 Loss 1.5514 1.512937358239746 Epoch 31 Loss 1.512954 Time taken for 1 epoch 0.55 sec	Epoch 41 Batch 0 Loss 1.4616 1.492629292275048 Epoch 41 Loss 1.492673 Time taken for 1 epoch 0.55 sec	Epoch 51 Batch 0 Loss 1.39817 1.451384046476968 Epoch 51 Loss 1.451358 Time taken for 1 epoch 0.77 sec	Epoch 61 Batch 0 Loss 1.3552 1.4292531339917 Epoch 61 Loss 1.429265 Time taken for 1 epoch 0.52 sec	Epoch 71 Batch 0 Loss 1.7921 1.436455656778157 Epoch 71 Loss 1.436456 Time taken for 1 epoch 0.53 sec	Epoch 81 Batch 0 Loss 1.4985 1.419149662922361 Epoch 81 Loss 1.419148 Time taken for 1 epoch 0.52 sec	Epoch 91 Batch 0 Loss 1.3948 1.377596804140973 Epoch 91 Loss 1.392896 Time taken for 1 epoch 0.47 sec	Epoch 101 Batch 0 Loss 1.3948 1.391127984261848 Epoch 101 Loss 1.391228 Time taken for 1 epoch 0.63 sec	Epoch 111 Batch 0 Loss 1.5852 1.377596804140973 Epoch 111 Loss 1.377596 Time taken for 1 epoch 0.47 sec	Epoch 121 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 121 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 131 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 131 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 141 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 141 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 151 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 151 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 161 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 161 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 171 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 171 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 181 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 181 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 191 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 191 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 201 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 201 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 211 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 211 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 221 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 221 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 231 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 231 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 241 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 241 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 251 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 251 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 261 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 261 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 271 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 271 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 281 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 281 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 291 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 291 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 301 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 301 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 311 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 311 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 321 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 321 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 331 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 331 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 341 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 341 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 351 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 351 Loss 1.371227 Time taken for 1 epoch 0.47 sec	Epoch 361 Batch 0 Loss 1.2019 1.3712267079561387 Epoch 361 Loss 1.371227 Time taken for 1 epoch 0.47 sec
---	---	--	--	--	---	--	--	--	--	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Figure 10: Epoch 1 to 36, Batch 0.

Epoch 37 Batch 0 Loss 0.4319 0.302515681873544 Epoch 37 Loss 0.302515 Time taken for 1 epoch 0.47 sec	Epoch 46 Batch 0 Loss 0.3935 0.3473659884828945 Epoch 46 Loss 0.347366 Time taken for 1 epoch 0.79 sec	Epoch 55 Batch 0 Loss 0.2582 0.2259621584728022 Epoch 55 Loss 0.225962 Time taken for 1 epoch 0.45 sec	Epoch 64 Batch 0 Loss 0.1847 0.189245311419109 Epoch 64 Loss 0.189245 Time taken for 1 epoch 0.46 sec	Epoch 38 Batch 0 Loss 0.5222 0.4041498618249835 Epoch 38 Loss 0.404149 Time taken for 1 epoch 0.47 sec	Epoch 47 Batch 0 Loss 0.4842 0.333418193715413 Epoch 47 Loss 0.333418 Time taken for 1 epoch 0.47 sec	Epoch 56 Batch 0 Loss 0.2180 0.2262626485967084 Epoch 56 Loss 0.226263 Time taken for 1 epoch 0.45 sec	Epoch 65 Batch 0 Loss 0.1728 0.1736618677739805 Epoch 65 Loss 0.173662 Time taken for 1 epoch 0.45 sec	Epoch 39 Batch 0 Loss 0.5295 0.375145428975421 Epoch 39 Loss 0.471515 Time taken for 1 epoch 0.47 sec	Epoch 48 Batch 0 Loss 0.2680 0.3167871541150411 Epoch 48 Loss 0.316787 Time taken for 1 epoch 0.46 sec	Epoch 57 Batch 0 Loss 0.2712 0.2151372631398896 Epoch 57 Loss 0.215137 Time taken for 1 epoch 0.46 sec	Epoch 66 Batch 0 Loss 0.1875 0.1681862311632282 Epoch 66 Loss 0.168186 Time taken for 1 epoch 0.62 sec	Epoch 40 Batch 0 Loss 0.5483 0.4478992484937744 Epoch 40 Loss 0.447898 Time taken for 1 epoch 0.48 sec	Epoch 49 Batch 0 Loss 0.4181 0.308395824475886 Epoch 49 Loss 0.308396 Time taken for 1 epoch 0.46 sec	Epoch 58 Batch 0 Loss 0.2631 0.21189368454646675 Epoch 58 Loss 0.211894 Time taken for 1 epoch 0.47 sec	Epoch 67 Batch 0 Loss 0.1618 0.16134792562629438 Epoch 67 Loss 0.161348 Time taken for 1 epoch 0.47 sec	Epoch 41 Batch 0 Loss 0.4543 0.4302839543384443 Epoch 41 Loss 0.430284 Time taken for 1 epoch 0.71 sec	Epoch 50 Batch 0 Loss 0.3181 0.29752204811546673 Epoch 50 Loss 0.297522 Time taken for 1 epoch 0.47 sec	Epoch 59 Batch 0 Loss 0.1875 0.19866821474282475 Epoch 59 Loss 0.198668 Time taken for 1 epoch 0.46 sec	Epoch 68 Batch 0 Loss 0.1622 0.1519527733259583 Epoch 68 Loss 0.151953 Time taken for 1 epoch 0.45 sec	Epoch 42 Batch 0 Loss 0.3514 0.4191981951395671 Epoch 42 Loss 0.419198 Time taken for 1 epoch 0.47 sec	Epoch 51 Batch 0 Loss 0.3275 0.2811824785841889 Epoch 51 Loss 0.281182 Time taken for 1 epoch 0.75 sec	Epoch 60 Batch 0 Loss 0.1776 0.285347577388146 Epoch 60 Loss 0.285348 Time taken for 1 epoch 0.46 sec	Epoch 69 Batch 0 Loss 0.1444 0.1451958837834672 Epoch 69 Loss 0.145189 Time taken for 1 epoch 0.47 sec	Epoch 43 Batch 0 Loss 0.4233 0.396558359275952 Epoch 43 Loss 0.395655 Time taken for 1 epoch 0.47 sec	Epoch 52 Batch 0 Loss 0.2687 0.2682555913925171 Epoch 52 Loss 0.268256 Time taken for 1 epoch 0.49 sec	Epoch 61 Batch 0 Loss 0.1889 0.19891196489334166 Epoch 61 Loss 0.198912 Time taken for 1 epoch 0.72 sec	Epoch 70 Batch 0 Loss 0.1297 0.1341850968295933 Epoch 70 Loss 0.134186 Time taken for 1 epoch 0.47 sec	Epoch 44 Batch 0 Loss 0.3358 0.3919767613983543 Epoch 44 Loss 0.381917 Time taken for 1 epoch 0.47 sec	Epoch 53 Batch 0 Loss 0.2945 0.25712819894154865 Epoch 53 Loss 0.257128 Time taken for 1 epoch 0.47 sec	Epoch 62 Batch 0 Loss 0.2091 0.2080929923855562 Epoch 62 Loss 0.208093 Time taken for 1 epoch 0.48 sec	Epoch 71 Batch 0 Loss 0.0967 0.138083684591268 Epoch 71 Loss 0.138088 Time taken for 1 epoch 0.73 sec	Epoch 45 Batch 0 Loss 0.4328 0.30831361822225 Epoch 45 Loss 0.368883 Time taken for 1 epoch 0.46 sec	Epoch 54 Batch 0 Loss 0.2533 0.248237538377875 Epoch 54 Loss 0.248224 Time taken for 1 epoch 0.46 sec	Epoch 63 Batch 0 Loss 0.1378 0.199745346889658 Epoch 63 Loss 0.199742 Time taken for 1 epoch 0.46 sec	Epoch 72 Batch 0 Loss 0.1871 0.111802827324799 Epoch 72 Loss 0.117883 Time taken for 1 epoch 0.46 sec
--	---	---	--	---	--	---	---	--	---	---	---	---	--	--	--	---	--	--	---	---	---	--	---	--	---	--	---	---	--	---	--	---	--	--	--

Figure 11: Epoch 37 to 72, Batch 0.

```

Epoch 73 Batch 0 Loss 0.0788 Epoch 82 Batch 0 Loss 0.0575 Epoch 92 Batch 0 Loss 0.0235
0.1072141428788583 Epoch 82 Loss 0.064931 0.050253347873687744
Epoch 73 Loss 0.107214 Epoch 82 Loss 0.064931 Epoch 92 Loss 0.058253
Time taken for 1 epoch 0.46 sec Time taken for 1 epoch 0.45 sec Time taken for 1 epoch 0.46 sec

Epoch 74 Batch 0 Loss 0.0981 Epoch 83 Batch 0 Loss 0.1123 Epoch 93 Batch 0 Loss 0.0485
0.10568176786631978 0.06196350355943044 0.06825476388136546
Epoch 74 Loss 0.105682 Epoch 83 Loss 0.061964 Epoch 93 Loss 0.068255
Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.47 sec

Epoch 75 Batch 0 Loss 0.0796 Epoch 84 Batch 0 Loss 0.0687 Epoch 94 Batch 0 Loss 0.0667
0.09788785985278523 0.05766955147480814 0.25376633803049725
Epoch 75 Loss 0.097887 Epoch 84 Loss 0.057670 Epoch 94 Loss 0.253766
Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.45 sec Time taken for 1 epoch 0.46 sec

Epoch 76 Batch 0 Loss 0.0736 Epoch 85 Batch 0 Loss 0.0406 Epoch 95 Batch 0 Loss 0.1257
0.09678077697753906 0.05540721615155538 0.24854536851247153
Epoch 76 Loss 0.096781 Epoch 85 Loss 0.055407 Epoch 95 Loss 0.248545
Time taken for 1 epoch 0.63 sec Time taken for 1 epoch 0.45 sec Time taken for 1 epoch 0.48 sec

Epoch 77 Batch 0 Loss 0.1027 Epoch 86 Batch 0 Loss 0.0591 Epoch 96 Batch 0 Loss 0.1939
0.0897419353261617 0.05089138749599457 0.3331497112909953
Epoch 77 Loss 0.089742 Epoch 86 Loss 0.050891 Epoch 96 Loss 0.333150
Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.63 sec Time taken for 1 epoch 0.79 sec

Epoch 78 Batch 0 Loss 0.0388 Epoch 87 Batch 0 Loss 0.0461 Epoch 97 Batch 0 Loss 0.2718
0.08615331848462422 0.048658753434816994 0.22792774438858032
Epoch 78 Loss 0.086153 Epoch 87 Loss 0.048659 Epoch 97 Loss 0.227928
Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.46 sec

Epoch 79 Batch 0 Loss 0.1057 Epoch 88 Batch 0 Loss 0.0475 Epoch 98 Batch 0 Loss 0.2661
0.07896246016025543 0.05038813749949137 0.25525617599487385
Epoch 79 Loss 0.078962 Epoch 88 Loss 0.050388 Epoch 98 Loss 0.255256
Time taken for 1 epoch 0.47 sec Time taken for 1 epoch 0.46 sec Time taken for 1 epoch 0.47 sec

Epoch 80 Batch 0 Loss 0.0899 Epoch 89 Batch 0 Loss 0.0657 Epoch 99 Batch 0 Loss 0.2079
0.07435195644695527 0.04787215776682427 0.21806711546579996
Epoch 80 Loss 0.074352 Epoch 89 Loss 0.047887 Epoch 99 Loss 0.210667
Time taken for 1 epoch 0.46 sec Time taken for 1 epoch 0.48 sec Time taken for 1 epoch 0.45 sec

Epoch 81 Batch 0 Loss 0.0286 Epoch 90 Batch 0 Loss 0.0458 Epoch 100 Batch 0 Loss 0.1956
0.06724690397580464 0.0467509138636851 0.1815354824066162
Epoch 81 Loss 0.067247 Epoch 90 Loss 0.046751 Epoch 100 Loss 0.181535
Time taken for 1 epoch 0.73 sec Time taken for 1 epoch 0.46 sec Time taken for 1 epoch 0.46 sec
    
```

Figure 12: Epoch 73 to 100, Batch 0.

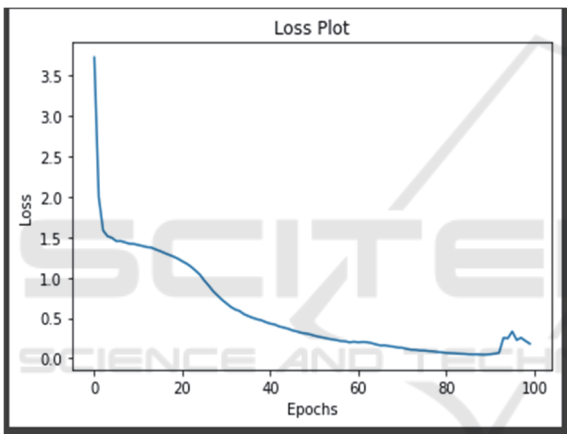


Figure 13: Comparison plot graph of label X for Epoch and label Y for Loss.

Epoch is an iteration with reverse propagation, based on previous research, the optimal number of epochs is influenced by various factors such as learning rate, optimizer, and amount of data. Based on the results of the study, the epochs used were 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. In this study, epochs 1 to 100 were used to obtain the largest total loss in epoch 1 of 3,581 and the smallest total loss. on epoch 91 is 0.0135. Based on Figures 13, 14 and 15 through the following epoch results, it can be seen that there is a correlation between the accuracy and loss values in the training data and the number of epochs or iterations. (A. Y. N. Richard Socher, Andrej Karpathy, 2014) The larger the epoch used, the higher the accuracy value on the data train. Inversely proportional to the accuracy value, the greater the epoch used, the lower the loss value generated in the training data. Based on this, it can be concluded that to reduce the loss value

obtained, it can be done by increasing the number of epochs in the training process, so that the model will produce a higher accuracy value. Based on the results of the validation test for accuracy and loss for 100 epochs, the following optimization was obtained.

Table 1: Comparison Table Accuracy and Loss Validation.

Epoch	Accuracy Validation	Loss Validation
10	1.417357	1.3940
20	1.234550	1.1343
30	0.726811	0.6227
40	0.447890	0.5403
50	0.297522	0.3181
60	0.205348	0.1776
70	0.134186	0.1297
80	0.074352	0.0899
90	0.046751	0.0458
100	0.181535	0.1956

Accuracy is a matrix to evaluate the results of the model classification. Accuracy is the division of the model's predictions that are considered correct with the predicted total. (D. Ariyoga, R. Rahmadi, and R. A. Rajagede, 2021).

6 CONCLUSION

Implementation of multi-task deep learning in video understanding to convert video into sentence text consists of processing stages, namely object and text area detection, determining Caption Region, Relation Region, and Object Region which then extracts features consisting of Caption Features, Relationship Features and Object Features and combine/process Relationship Features, Caption Features and Context Relation Features with the CCN (Caption Context Network) method and the RCN (Relationship Context Network) method. The accuracy results obtained for classifying accuracy validation against loss are obtained from the results of research with 100 epochs, the largest total loss is obtained in epoch 1 of 3.581 and the smallest total loss is at epoch 91, which is 0.0135.

REFERENCES

L. R. Jácome-Galarza, M. A. Realpe-Robalino, L. A. Chamba-Eras, M. S. Viñan-Ludeña, and J. F. Sinche-Freire, "Computer Vision for Image Understanding: A Comprehensive Review," Adv. Intell. syst. Comput.,

- vol. 1066, no. September 2020, pp. 248–259, 2020, doi: 10.1007/978-3-030-32022-5_24.
- S. Bai et al., “Natural language guided visual relationship detection,” *Math. problem. Eng.*, vol. 2020, no. 1, pp. 444–453, 2018, doi:10.1145/3219819.3220036.
- M. Sundermeyer, R. Schlüter, and H. Ney, “LSTM neural networks for language modeling,” 2012.
- O. Sener and V. Koltun, “Multi-task learning as multi-objective optimization,” 2018.
- X. Liu, P. He, W. Chen, and J. Gao, “Multi-task deep neural networks for natural language understanding,” arXiv, pp. 4487–4496, 2019.
- A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” 2016.
- A. Karpathy and L. Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 664–676, 2017, doi:10.109/TPAMI.2016.2598339.
- R. Staniute and D. ešok, “A systematic literature review on image captioning,” *Appl. Sci.*, vol. 9, no. 10, 2019, doi:10.3390/app9102024.
- D. Shin and I. Kim, “Deep Image Understanding Using Multilayered Contexts,” *Math. problem. Eng.*, vol. 2018, 2018, doi:10.1155/2018/5847460.
- S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi:10.109/TPAMI.2016.2577031.
- L. R. Jácome-Galarza, M. A. Realpe-Robalino, L. A. Chamba-Eras, M. S. Viñán-Ludeña, and J. F. Sinche-Freire, “Computer Vision for Image Understanding: A Comprehensive Review,” *Adv. Intell. syst. Comput.*, vol. 1066, no. May, pp. 248–259, 2020, doi: 10.1007/978-3-030-32022-5_24.
- S. Bai and S. An, “A survey on automatic image caption generation,” *Neurocomputing*, vol. 311, pp. 291–304, 2018, doi:10.1016/j.neucom.2018.05.080.
- S. Aditya, Y. Yang, C. Baral, Y. Aloimonos, and C. Fermüller, “Image Understanding using vision and reasoning through Scene Description Graph,” *Comput. vis. Image Underst.*, vol. 173, no. December, pp. 33–45, 2018, doi:10.1016/j.cviu.2017.12.004.
- A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” 33rd Int. conf. Mach. Learn. ICML 2016, vol. 4, pp. 2611–2620, 2016.
- A. Y. N. Richard Socher, Andrej Karpathy, Quoc V. Le*, Christopher D. Manning, “Grounded