**UNIVERSITY OF PIRAEUS**

**School of Finance and Statistics**



Graduate Study Program in

**Advanced Informatics and Computing Systems - Software Development and Artificial Intelligence**

**English Sign Language recognition from video, using Convolutional-Recurrent Neural Network and Deep Sequence Models**

by

MARTHA TODI

Supervisor Professor: **Dionysios Sotiropoulos**

Bachelor’s Thesis

Submitted to the Department of Informatics

Of the University of Piraeus

in partial fulfilment of the requirements for the master’s degree in Advanced Informatics and Computing Systems - Software Development and Artificial Intelligence

**Piraeus**

**November 2022**

*To my parents*

*Dimitris and Kristallenia*

Thanks to …

**Abstract**

Hearing-impaired people are using Sign Language (SL) as their main communication with other people. This language contains mostly the combined features of hand motions and facial expressions to define the words and the sentences. The relation between the whole meaning of a sentence and the specific words is -in some cases- complicated and hard to understand by Deaf Community. Also, has to be mentioned the fact that people with visual difficulties don’t have any way to communication directly with moo people. That is the reason that the translation must be automatically. The process to achieve that can be made from AI and machine learning and right now the research community finds difficulties, because of the lack of large datasets and the differences of each language. Therefore, the need to help deaf community in their communications is big and engineers are trying to find methods and ways to establish it. One of the first issues is the motion recognition of a human that communicate in SL by the machine. Another issue is to match the hands motions with the word. The main way is by training Artificial Neural Networks and improving continuously every matter that come across. So, using a lot of different approaches and testing the conclusions is crucial. This thesis is trying to reach a small goal in this big vision and help other researcher to translate SL in other languages, with the help of image classification and Deep Sequence Models

**Contents**

1.Introduction7

**2.Introduction into Deep Sequence Models for image classification8**

2.1 Image classification 5

2.2 Convolutional Neural Networks 5

2.3 Sequence Models 5

2.4 Deep Sequence Models 5

2.4.1 Recurrent Neural Network 5

2.4.2 Transformer-based model  5

**3.Relevant literature on sign language recognition 7**

**3.1 Sensor Placement for Bi-Channel recognition system proposed by Kim5**

**3.2 Classification of low-level surface electromyogram5**

**3.3 SL Recognition based on Intelligent Gloves Using ML techniques 5**

**4.Technical Environment7**

**5.Dataset Description7**

**5.1 WLASL** **5**

**3.1 Label’s Preparation5**

**6. Data Preprocessing7**

**7. Model Architecture7**

**8. Experimental Results7**

**9. Discussion7**

**10. Future Work7**

**11. Bibliography7**

**Catalog of pictures**

**CHAPTER 1**

**Introduction**

Artificial Neural Networks (ANNs), serving the evolution of Machine Learning (ML) and by extension Artificial Intelligence (AI), are used in more and more fields and their contribute to research for the acquisition of specialized knowledge are increasing. The same applies to diverse applications in information systems that we use every day. One of their usages is in the translation of words between different languages.

Their use does not only concern words but - among other things - also images and audio-visual data. The translation of SL into speaking language certainly cannot be achieved through written language. Observation of human movement, either physically or via video, is required. This thesis involves the neural system with the process of deep learning algorithms and by correlating minute-long sign language (ESL) videos corresponding to one English language word. The purpose of the project is to train the system resulting in the recognition of the word in videos where a human reproduces it in sign language. For the success of the result, the image classification method was applied to several series of image (frames) of each video and analyzing by Deep Sequence Models.

**Chapter 2**

**Introduction into Deep Sequence Models for image classification**

The recognition of human movement in video by neural networks is done by classifying each situation for the analysis and extraction of the information from the movement. The training of the neuron with this process is called image classification(IC) n. A video, however, is a continuous situation over time and it is necessary to record and store it, in order to fully understand the action presented. Sequence Models(SM) have been motivated by this scope.

Therefore, the hybrid architecture of deep sequence Models with image classification is quite effective because is needed the spatial information of IC tasks and the temporal from the SM.

**2.1 Image classification**

An image can convert to groups of pixels or vectors, as well the multiple frames of a video. Image classification is the task of convolutional neural networks (CNN) that categorize and adding labels to them, depending on specific rules and finding same features. The scope is to recognize what an image represents.

An example of one of the most typical image classifications is face recognize.

For this thesis, the first need is to classify the movement for each world of sign language and localize only the human’s hands of the whole picture (spatial processing).

The videos are used for the input layer of the NN and after the process of hidden layers with image classification produce the percentage quantity of the words that are representing.

<https://viso.ai/computer-vision/image-classification/>

# 2.2 Convolutional Neural Networks(CNN )

A CNN is an architecture for deep learning that training directly on data. The scope is to find patterns in frames to recognize specific objects or classify series of video’s frames.

In order to achieve image classification, it is necessary to introduce a CNN layer, which will separate the image into pieces based on filters, which determine the size of the focused area.

# <https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4>

# Feature Extraction is the part where relevant information from the data is taken and enhanced. It cuts out the redundant information from the region of interest and start taking the features to be used for the classifier calculation. The most common feature extraction is by using the convolution layers of Convolutional Neural Network (CNN)

# https://www.sciencedirect.com/science/article/pii/S1877050921000442

Sequence Models have been motivated by the analysis of sequential data such text sentences, time-series and other discrete sequences data. These models are especially designed to handle sequential information while Convolutional Neural Network are more adapted for process spatial information.(change that a little bit)

<https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20CNN%20typically%20has%20three,and%20a%20fully%20connected%20layer>.

If the frame of a video -as input- consists of multiple image inputs as the first layer, which features can be extracted in the next layer. This layer is a whole CNN model that followed by RNN model – layer to get the sequent data and produced the result of image recognition.

An image of these example shown in Figure2.

Diagram

Description automatically generated

Figure 1

Neural Network Based Image Captioning

https://link.springer.com/chapter/10.1007/978-3-030-04780-1\_23

**2.3 Sequence Models**

The training data of NNs can be audio clip, text streams or video clips that represent time sequences of data. As we mentioned, the training videos for this thesis are consists of sequence of video frames. Therefore, the model relates the information across different timestamps, so that can predict the future time series or to recognize speech/voice or process natural language .

Sometimes both the input and output are sequences, in some either the input or the output is a sequence (change this line)

Diagram

Description automatically generated

The types of these models are 3. Many to one (e.g. sentiment classification), One to Many(e.g. image captioning) and Many to Many(e.g. Machine translation) and there is another Many to Many that includes Video Classification on frame level as shown on Figure 1

Figure https://deeplearningmath.org/sequence-models.html

**2.4 Deep Sequence Models**

**2.4.1 Recurrent Neural Network**

Recurrent neural network (RNN) is a popular sequence model that has shown efficient performance for sequential data as in that case is video.

<https://www.techtarget.com/searchenterpriseai/feature/CNN-vs-RNN-How-they-differ-and-where-they-overlap#:~:text=RNNs%20are%20better%20suited%20to,more%20on%20this%20point%20below>).

The main advantage of using RNNs instead of standard neural networks is that the features are not shared in standard neural networks. Weights are shared across time in RNN. RNNs can remember its previous inputs but Standard Neural Networks are not capable of remembering previous inputs. RNN takes historical information for computation.

<https://towardsdatascience.com/sequence-models-and-recurrent-neural-networks-rnns-62cadeb4f1e1>

**RNN** can tackle the following challenges from sequence data:

* Dealing with variable-length sequences
* Maintain sequence order
* Keep track of long-term dependencies
* Share parameters across the sequence

<https://deeplearningmath.org/sequence-models.html>

https://keras.io/guides/working\_with\_rnns/

**Recurrent Neural Networks**are very powerful, because they combine 2 properties: 1) distributed hidden state that allows them to store a lot of information about the past efficiently, and 2) non-linear dynamics that allow them to update their hidden state in complicated ways. With enough neurons and time, RNNs can compute anything that can be computed by your computer. So what kinds of behavior can RNNs exhibit? They can oscillate, they can settle to point attractors, they can behave chaotically. And they could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects.

https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html

**2.4.2. Transformer-based model**

As Fred Navruzov said "Nowadays, the boundaries between CNN and RNN usage are somewhat blurred, as you can combine those architectures into CRNN for increased effectiveness in solving specific tasks like video tagging or gesture recognition,"

A transformer is a deep learning model that gives the ability of attention to the training process as it can enhances some parts of the input data while diminishing other parts -as called the mechanism of self-attention. This advantage can be exceeded with different weights on the dividing input data. In this case, the hands will have highest weights, while the wall behind the human zero

Like RNNs, transformers are designed to process sequential input data, such as natural language, with applications towards tasks such as translation and text summarization. However, unlike RNNs, transformers process the entire input all at once. The attention mechanism provides context for any position in the input sequence and make easy to work parallel, capture the whole image, and eventually to decide which part is important. That is one difference with RNNs.

**https://en.wikipedia.org/wiki/Transformer\_(machine\_learning\_model)#:~:text=A%20transformer%20is%20a%20deep,and%20computer%20vision%20(CV).**

The transformer neural network receives an input sentence and converts it into two sequences: a sequence of word vector embeddings, and a sequence of positional encodings.

The word vector embeddings are a numeric representation of the text. It is necessary to convert the words to the embedding representation so that a neural network can process them. In the embedding representation, each word in the dictionary is represented as a vector. The positional encodings are a vector representation of the position of the word in the original sentence.

**https://deepai.org/machine-learning-glossary-and-terms/transformer-neural-network#:~:text=What%20is%20a%20Transformer%20Neural,area%20of%20natural%20language%20processing.**

**NVIDIA**

**https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/**

**Chapter 3**

**3. Relevant literature on sign language recognition**

On the past years, ‘real’ machine visual was very difficult to achieved. Therefore, the ways that replaced it had more hardware equipment to capture hands’ motions using appropriate sensing devices in marker-less or marker-based setups. The sensors that applied to those devices have very difficult work to do and the and so the accuracy in their metrics is limited because of resolution and discrimination ability. That is the reason why researchers reach another approach of development deep learning algorithms to optimize machine visual for SL recognition.

Nonetheless, there are some very good projects that has succeed a big part of this goal and an introduction of some of them is following.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8434597/>

* 1. **Sensor Placement for Bi-Channel recognition system proposed by Kim**

In this project in order to capture the human movement and more specific hand’s motion had been used the accelerometer (ACC) and electromyogram (EMG) as shown on Figure3.

The accelerometer used in experiment provides the rate of change of velocity along three axes (x, y, z). For the analysis, each of the three channels was treated separately.

Commonly, the EMG signal requires additional pre-

processing such as deep smoothing depending on the po-

sition of the sensor, because the nature of the signal is such

that all the muscle ﬁbers within the recording area of the

sensor contract at different rates

As the EMG signal requires additional pre-processing such as deep smoothing depending on the position of the sensor, because the nature of the signal is such that all the muscle ﬁbers within the recording area of the sensor contract at different rates

Using the most relevant 15 features, an average accu-

racy of 99.82% is achieved for subject-dependent recogni-

tion. The universality of the selected features is proven for

all subjects with an average accuracy of 96.31%. Unfortu-

nately, this high accuracy does not carry over to the subject-

independent recognition.

Using the most relevant 15 features, an average accuracy of 99.82% is achieved for subject-dependent recognition. The universality of the selected features is proven for all subjects with an average accuracy of 96.31%. Unfortunately, this high accuracy does not carry over to the subject-independent recognition.

In this paper, we investigate the potential of two sensors,

accelerometer and electromyogram (EMG), for differentiat-

ing word-level sign vocabularies in German Sign Language

(GSL). The main goal of this work is to examine the com-

plementary functionality of both sensors in sign language

recognition and to determine an efﬁcient fusion scheme for

bi-channel sensor combination. Because of the character-

istics of the sensors that measure motion and muscle con-

traction in a synchronized time scale and single dimension,

we focus mainly on the feature-level fusion to classify bi-

channel features and discuss a problem of feature-level fu-

sion caused by high disparity between accuracies of each

single channel classi

In this paper, we investigate the potential of two sensors,

accelerometer and electromyogram (EMG), for differentiat-

ing word-level sign vocabularies in German Sign Language

(GSL). The main goal of this work is to examine the com-

plementary functionality of both sensors in sign language

recognition and to determine an efﬁcient fusion scheme for

bi-channel sensor combination. Because of the character-

istics of the sensors that measure motion and muscle con-

traction in a synchronized time scale and single dimension,

we focus mainly on the feature-level fusion to classify bi-

channel features and discuss a problem of feature-level fu-

sion caused by high disparity between accuracies of each

single channel classiﬁcation.

In this paper, we investigate the potential of two sensors,

accelerometer and electromyogram (EMG), for differentiat-

ing word-level sign vocabularies in German Sign Language

(GSL). The main goal of this work is to examine the com-

plementary functionality of both sensors in sign language

recognition and to determine an efﬁcient fusion scheme for

bi-channel sensor combination. Because of the character-

istics of the sensors that measure motion and muscle con-

traction in a synchronized time scale and single dimension,

we focus mainly on the feature-level fusion to classify bi-

channel features and discuss a problem of feature-level fu-

sion caused by high disparity between accuracies of each

single channel classiﬁcation.

Diagram

Description automatically generated

Figure 3 Sensor placement. <https://www.researchgate.net/publication/221292288_Bi-channel_sensor_fusion_for_automatic_sign_language_recognition>

A picture containing indoor, weapon

Description automatically generated

**3.2Classification of low-level surface electromyogram using independent component analysis by**

Figure 4 Hand gesture experimental set up with four electrodes

A picture containing text, indoor

Description automatically generatedThe experiments have compared the accuracy of identifying the hand actions based on the RMS of SEMG recordings using raw SEMG, SEMG separated using standard ICA and using the unmixing matrix and weight matrix corresponding to the individual. These results have been tabulated in Table 3. From this table, it is observed that classification of SEMG after pre-processing using the unmixing matrix and corresponding weight matrix has 97% accuracy. This accuracy is 65% when only the unmixing matrix is generated for each set of experiments. When RMS from unseparated SEMG was used (referred to as raw SEMG), the accuracy of classification was only 60%

**https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8749583/**

**3.3 SL Recognition based on Intelligent Gloves Using ML techniques**

The present work presents a ﬁrst approach an intelligent glove to determine the best methodology to ﬁnd the best training set (less data as possible) from sensor data in electronic system. the data are acquired from glove on the right hand used by a deaf person to raise information of the numbers between 1 to 10 in SL. To determinate each SL number a k-NN algorithm is implemented as classiﬁer.

The proposed methodology met the objective of reducing the greater amount of data in the training set and that when implementing a classiﬁer, it has a high performance. In this way it is possible to store all the alphabet in sign language

A picture containing indoor, bag, backpack, handwear

Description automatically generated

[**https://www.researchgate.net/figure/Hand-gesture-experimental-set-up-with-four-electrodes\_fig2\_224182799**](https://www.researchgate.net/figure/Hand-gesture-experimental-set-up-with-four-electrodes_fig2_224182799)

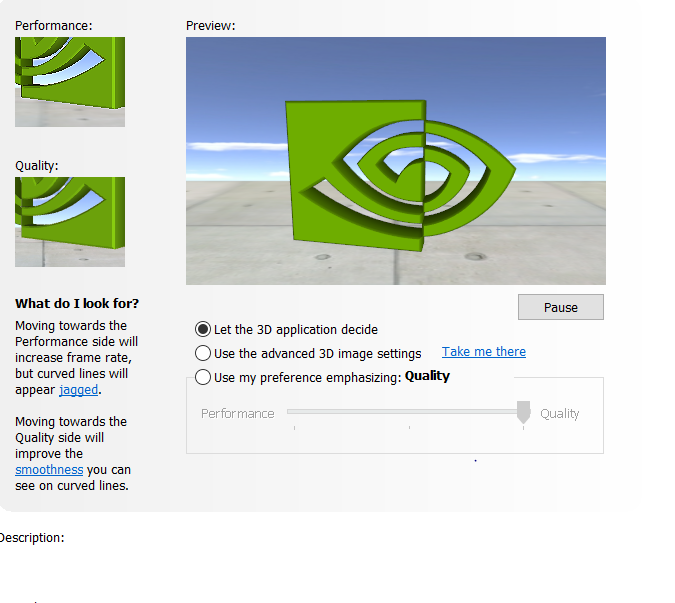
[**https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.736.1160&rep=rep1&type=pdf**](https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.736.1160&rep=rep1&type=pdf)

**Chapter 4**

**Technical Environment**

Machine Learning and specially Technical Neural Networks needs a strong technical environment.

The machine that project is running has NVIDIA GEFORCE GTX 1650 graphic card .



CUDA (Compute Unified Device Architecture) is a [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing) platform from Nvidia Corporation. Provides and API that allows software to use certain types of [graphics processing units](https://en.wikipedia.org/wiki/Graphics_processing_units) (GPUs) for general purpose processing, an approach called general-purpose computing on GPUs. CUDA is a software layer that gives direct access to the GPU's virtual [instruction set](https://en.wikipedia.org/wiki/Instruction_set) and parallel computational elements, for the execution of [compute kernels](https://en.wikipedia.org/wiki/Compute_kernel).[[1]](https://en.wikipedia.org/wiki/CUDA#cite_note-CUDA_intro_-_TomsHardware-1)\

ANACONDA

The code of the project is written in Python. Anaconda is a good solution to develop machine learning especially in one machine .

Conda

Conda is an open source package management system and environment management system that runs on Windows, macOS, Linux and z/OS. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language.Conda as a package manager helps you find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

<https://docs.conda.io/en/latest/>

KERAS

Keras is a deep learning API written in Python, that runs on top of the machine learning platform Tensorflow. It was developed with a focus on enabling fast experimentation.

https://keras.io/about/

Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

https://en.wikipedia.org/wiki/Keras

This project built on Keras because supports CNN and RNN. It supports other common utility layers that is used and explained later , like dropout and pooling .

TENSORFLOW 2.5

TensorFlow is a free end-to-end and open-source software platform that this project using it for training and inference for machine learning and AI .It is an infrastructure layer for [differentiable programming](https://en.wikipedia.org/wiki/Differentiable_programming). It combines four key abilities:

* Efficiently executing low-level tensor operations on CPU, GPU, or TPU.
* Computing the gradient of arbitrary differentiable expressions.
* Scaling computation to many devices, such as clusters of hundreds of GPUs.
* Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices.

Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

This project imports NN frameworks that Tensorflow provides, such as Pytoch and Nympy .

**Chapter 5**

**5. Data set Description**

The idea of this thesis would never been held if the amazing Database of WLASL isn’t exists.

WLASL is the largest video dataset for Word-Level American Sign Language (ASL) recognition, which features 2,000 common different words in ASL . This dataset is Licensed under the Computational Use of Data Agreement (C-UDA).

<https://dxli94.github.io/WLASL/>

**5.1 WLASL**

Research for machine learning requires as many as possible data set for training. Especially on video classification , where videos have a lot of information , the size of dataset must be at least 2000 videos. The WLASL provides 8000 small videos, among 1 and 3 sec, and each of them corresponds in one word. So, every of the 2000 different words, are being described by 4 videos, which presents different people, with different background each time. This gives the ability to model to train the neural with more accuracy and to provides a good sample, for recognizing the desirable part of the video and to develop more chances for succeed prediction,

To clone it from git <https://github.com/dxli94/WLASL.git>

<https://github.com/dxli94/WLASL>

For download needs the youtube-dll (<https://github.com/ytdl-org/youtube-dl>)

To running on Linux.

Graphical user interface, text, application

Description automatically generated

Figure licence of the DB

<https://paperswithcode.com/dataset/wlasl>

**5.2 Labels preparation**

Inside the downloaded file exists a folder where all of the videos has not a name with the world that represents .

Graphical user interface, website

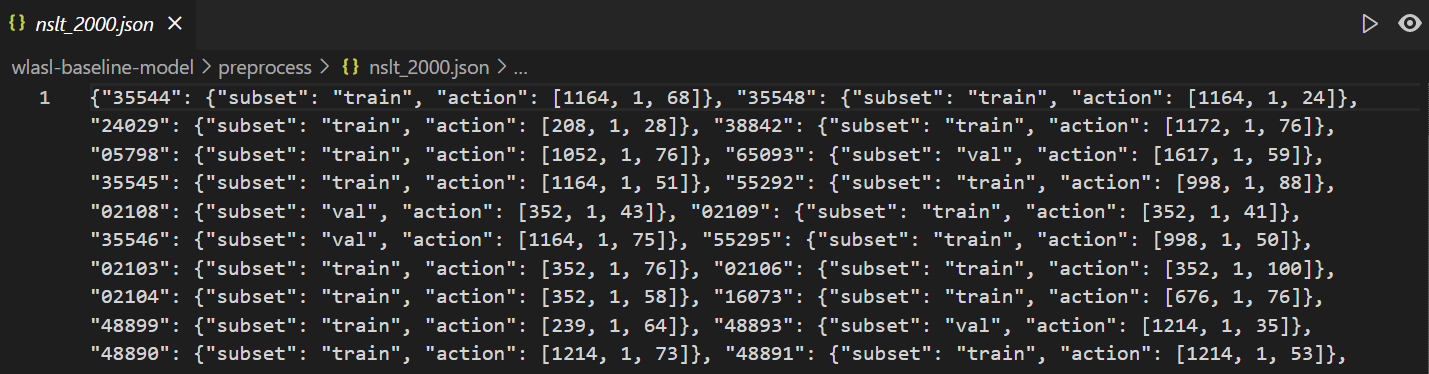
Description automatically generated

Also there is a file wlasl\_class\_list.txt

Text

Description automatically generated with low confidence

And a nslt\_200.json



Where in action the third value is the number of wlasl\_class\_list.

For this reason, has to separate and extract video’s names with their corresponding labels . Running the code below,

**import** **json**

i=0

j=0;

d=''

must=[]

data={}

datafirst={}

datasec={}

**with** open('preprocess/nslt\_2000.json', 'r') **as** r:

element = json.load(r)

**for** v **in** list(element):

datafirst.update({v:element[v]['action'][0]})

**with** open('preprocess/wlasl\_class\_list.txt','r') **as** f:

line = f.readline()

**for** line **in** f:

action=line.split()[0]

translate=line.split()[1]

datasec.update({action:translate})

**for** d **in** datafirst:

**for** s **in** datasec:

*#print(datafirst[d])*

*#print(s)*

i=i+1

j=j+1

**if**(str(s)==str(datafirst[d])):

data.update({d:datasec[s]})

print(data)

Gives the output of the image above which is the desirable json .

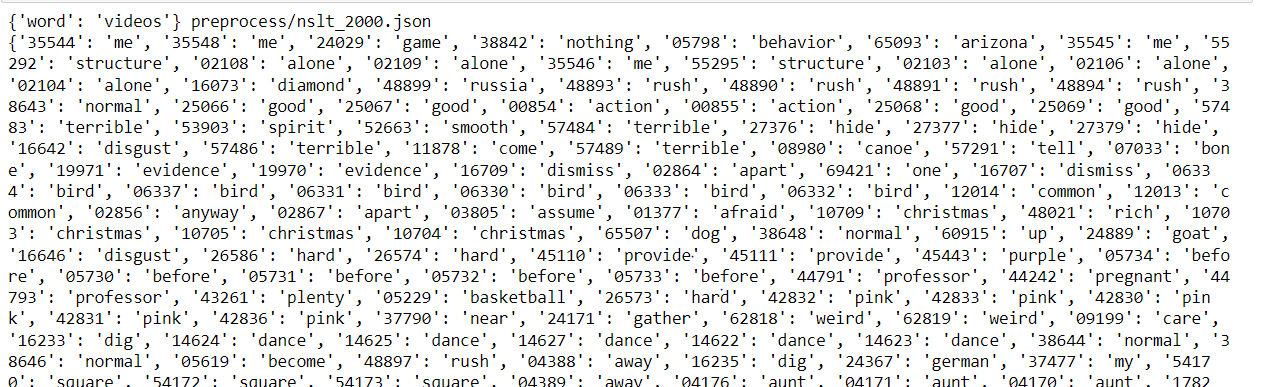
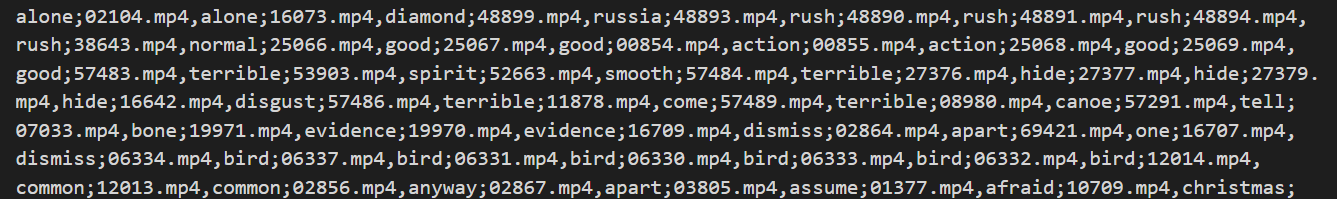


Figure The output of json

The final file that must create to be readable from the model has to be csv formatted . So, replacing the ‘:’ with comma and the comma -which separates the worlds – with ‘;’ , converts the data like the image above



Which can be converted to .csv file which contains rows with the name of the videos and their labels

Graphical user interface, application, table, Excel

Description automatically generated

**Chapter 6**

**6. Data preprocessing**

The layers that exist in this project takes data that must be preprocessed, so that the format can be available to the training model. Especially for a dataset of videos which are ordered sequence of frames and must extract the frames and add them in a 3D tensor. As the size of a video may be differ, so the number of frames changes between each video. The solution is to save the frames of videos at a fixed interval depend on a maximum frame count. If the extracted frames is lesser than this count, the video will be padded with zeros.

**6.1. Hyperparameters**

First, must define the hyperparameters to:

MAX\_SEQ\_LENGTH = 20 frames must contain each video.

NUM\_FEATURES = 2048

IMG\_SIZE = 224.

BATCH\_SIZE = 64

EPOCHS = 10

**6.2. Build Feature Extractor**

The extracted frames have some features that are not meaningful for the training. To extract only the useful parts an InceptionV3 model from Keras Applications has been used. This is a pre-trained network which returns a Keras image classification model.

Inception Architecture can lead to high performance vision networks that have a relatively modest computation cost compared to simpler, more monolithic architectures

https://arxiv.org/pdf/1512.00567.pdf

Inception v3 is an image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset.

Before the model can be used to recognize images, it must be trained using a large set of labeled images. ImageNet is a common dataset to use.

https://cloud.google.com/tpu/docs/inception-v3-advanced

The model that created is a fully-connected layer at the top, as the last layer of the network with weights of a pretraining on ImageNet, with three (3) Optional shape tuple input ,which width and heigh is 224 and gives a 2D output . As the code that is used shows bellow

def build\_feature\_extractor():

feature\_extractor = keras.applications.InceptionV3(

weights="imagenet",

include\_top=False,

pooling="avg",

input\_shape=(IMG\_SIZE, IMG\_SIZE, 3),

)

preprocess\_input = keras.applications.inception\_v3.preprocess\_input

inputs = keras.Input((IMG\_SIZE, IMG\_SIZE, 3))

preprocessed = preprocess\_input(inputs)

outputs = feature\_extractor(preprocessed)

return keras.Model(inputs, outputs, name="feature\_extractor")

feature\_extractor = build\_feature\_extractor()

What will be returned from the layers that has been grouped into an object , is a Functional API model

(image apo ta apotelesmata successful and could not)

**6.2.1 Crop Center Square**

After extracting the rows and the columns values from each frame , and make calculations which taken from the git code https://github.com/tensorflow/hub/blob/master/examples/colab/action\_recognition\_with\_tf\_hub.ipynb , returns the cropped frame.

def crop\_center\_square(frame):

y, x = frame.shape[0:2]

min\_dim = min(y, x)

start\_x = (x // 2) - (min\_dim // 2)

start\_y = (y // 2) - (min\_dim // 2)

return frame[start\_y : start\_y + min\_dim, start\_x : start\_x + min\_dim]

**6.3 Video Capture**

To read frames from videos the OpenCV’s VideoCapture method.

VideoCapture is a function of openCV library(used for computer vision, machine learning, and image processing) which allows working with video either by capturing via live webcam or by a video file.

**https://www.geeksforgeeks.org/how-to-get-properties-of-python-cv2-videocapture-object/**

The VideoCapture class returns **a video capture object** which we can use to display the video.

https://dontrepeatyourself.org/post/how-to-read-and-write-videos-with-opencv/

def load\_video(path, max\_frames=0, resize=(IMG\_SIZE, IMG\_SIZE)):

cap = cv2.VideoCapture(path)

frames = []

try:

while True:

ret, frame = cap.read()

if not ret:

break

frame = crop\_center\_square(frame)

frame = cv2.resize(frame, resize)

frame = frame[:, :, [2, 1, 0]]

frames.append(frame)

if len(frames) == max\_frames:

break

finally:

cap.release()

return np.array(frames)

**6.4 StringLookUp**

The labels of the videos are strings. Neural networks do not understand string values, so they must be converted to some numerical form before they are fed to the model. Here we will use the [StringLookup](https://keras.io/api/layers/preprocessing_layers/categorical/string_lookup) layer encode the class labels as integers.

<https://keras.io/examples/vision/video_classification/>

This layer will cause an error if even one token that used is out of vocabulary and it takes a 1D array that contains all of the translations of videos for train. The output will be the converted input strings to their index in the vocabulary .

label\_processor = keras.layers.StringLookup(

num\_oov\_indices=0, vocabulary=np.unique(train\_df["tag"])

)

print(label\_processor.get\_vocabulary())

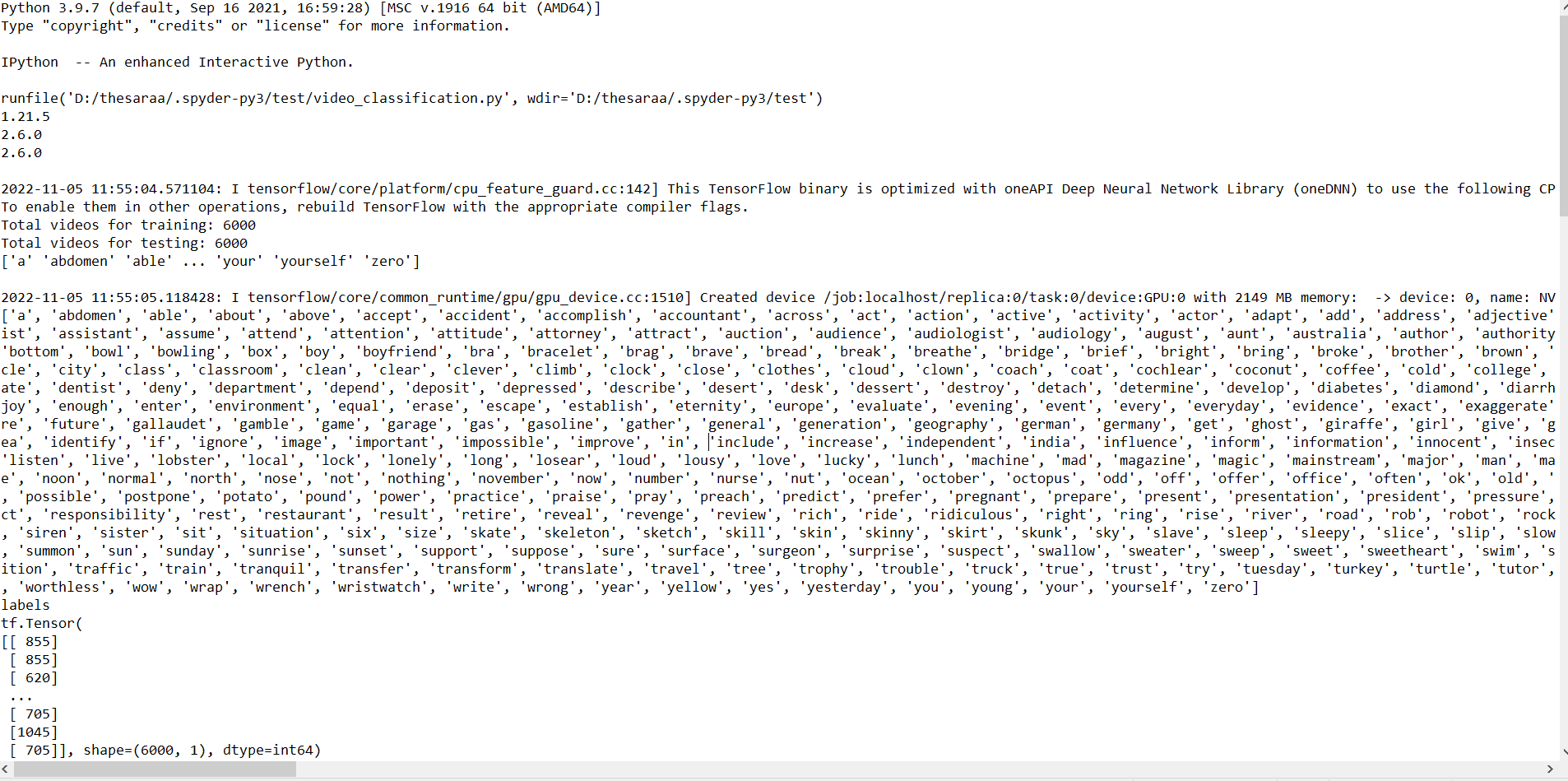
**6.5 Prepare All Videos**

In the end , calling all the above

Text

Description automatically generated

Which output is bellow.



A picture containing text

Description automatically generated

Graphical user interface, application

Description automatically generated

Now the data is ready for use.

**Chapter 7**

**7. Model Architecture**

The dataset for training that have used are videos of 3 seconds max duration. For the purpose of successful image classification, each video has to split into frames that are connected in an ordered sequence. A frame contains 2D pictures full of information about the location that each data has, compared with other data inside those pictures. Moreover, the ordered sequence frames contain 1D(time) locality. Therefore, two kinds of very useful information arise:

1. Spatial Information
2. Temporal Information

To model both of these aspects, we use a hybrid architecture that consists of convolutions (for spatial processing) as well as recurrent layers (for temporal processing).

<https://keras.io/examples/vision/video_classification/>

As mentioned in the previous chapter, the spatial processing can be used from Convolutional Neural Network and the temporal processing from Recurrent Neural Network.

So, it is not hard to understand why the current hybrid architecture is called CNN-RNN.

Specifically, we'll use a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) consisting of GRU layers. This kind of hybrid architecture is popularly known as a **CNN-RNN**.

**7.1 Layers Schematic**

**7.2 Set Up**

import numpy

print(numpy.version.version)

import tensorflow as tf

print(tf.\_\_version\_\_)

from tensorflow\_docs.vis import embed

from tensorflow import keras

print(keras.\_\_version\_\_)

from imutils import paths

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import imageio

import cv2

import os Torch

PyTorch

In machine learning, when we represent data, we need to do that numerically. A tensor is simply a container that can hold data in multiple dimensions.

Pandas

Is a library for data manipulation and [analysis](https://en.wikipedia.org/wiki/Data_analysis).[[2]](https://en.wikipedia.org/wiki/Pandas_(software)#cite_note-2) In particular, it offers [data structures](https://en.wikipedia.org/wiki/Data_structure) and operations for manipulating numerical tables and [time series](https://en.wikipedia.org/wiki/Time_series) and is built upon Numpy.

Nympy

NumPy is a Python library used for working with arrays. In this project using zeros Return a new array of given shape and type, filled with zeros.

**7.3 The Sequence Model**

**7.3.1 GRU**

For the problem of the short-term memory first, two GRU layers are created.

Based on available runtime hardware and constraints, this layer will choose different implementations (cuDNN-based or pure-TensorFlow) to maximize the performance. If a GPU is available and all the arguments to the layer meet the requirement of the cuDNN kernel (see below for details), the layer will use a fast cuDNN implementation.

https://keras.io/api/layers/recurrent\_layers/gru/

The first layer has 16 dimensionality of the output space and will return the last output in the output sequence. Its input is a symbolic 20-shaped tensor-like object and the corresponding timestep will be utilized. <https://keras.io/api/layers/recurrent_layers/gru/>

This layer will be the input of another GRU layer, with the half dimensionality

frame\_features\_input = keras.Input((MAX\_SEQ\_LENGTH, NUM\_FEATURES))

mask\_input = keras.Input((MAX\_SEQ\_LENGTH,), dtype="bool")

x = keras.layers.GRU(16, return\_sequences=True)(

frame\_features\_input, mask=mask\_input

)

x = keras.layers.GRU(8)(x)

<https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6>

**7.3.2 Dropout Layer**

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.

<https://keras.io/api/layers/regularization_layers/dropout/>

This layer takes as input the GRU last layer with a 0.4 fraction of its units to drop.

x = keras.layers.Dropout(0.4)(x)

Keras Dense

The dense layer is a neural network layer that is connected deeply, which means each [neuron](https://machinelearningknowledge.ai/glossary/artificial-neuron/) in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The output generated by the dense layer is an ‘m’ dimensional vector. Thus, dense layer is basically used for changing the dimensions of the vector.

<https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/#:~:text=The%20dense%20layer%20is%20a,performs%20a%20matrix%2Dvector%20multiplication>.

In the project the number of a layer’s outputs is 8, using a Relu activation function in the dense layer which is hidden. This output is the input of another hidden layer , which its size is the length of the formatted via StringLookUp and get\_vocabulary labels of the training videos.

x = keras.layers.Dense(8, activation="relu")(x)

output = keras.layers.Dense(len(class\_vocab), activation="softmax")(x)

**7.3.3 RNN model**

In the end of building the sequence model taking the output of the last hidden layer .

rnn\_model = keras.Model([frame\_features\_input, mask\_input], output)

The training process configuring with compile() , with a crossentropy loss function,which returns a floar tensor and the labels provided as integers . The optimizer of Adam algorithm. Also , the metrics have calculations of the frequency where the predictions equal the labels .

rnn\_model.compile(

loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"]

)

return rnn\_model

**7.3.4 Run Experiment**

First , the callback for ModelCheckpoint saves ,the latest best in quantity, model’s weights in a checkpoint file to be loaded for the training , while showing updates of the starting action. As shwn the img

filepath = "/tmp/video\_classifier"

checkpoint = keras.callbacks.ModelCheckpoint(

filepath, save\_weights\_only=True, save\_best\_only=True, verbose=1

)

seq\_model = get\_sequence\_model()

The training starts with the fit() which has as input the first 2 columns of trained data and the target is the train\_labels, with a 0.3 franction to the last samples, to evaluate the loss and the metrics in each of the 10 epoch.

history = seq\_model.fit(

[train\_data[0], train\_data[1]],

train\_labels,

validation\_split=0.3,

epochs=EPOCHS,

callbacks=[checkpoint],

)

seq\_model.load\_weights(filepath)

Showing the accuracy of each epoch

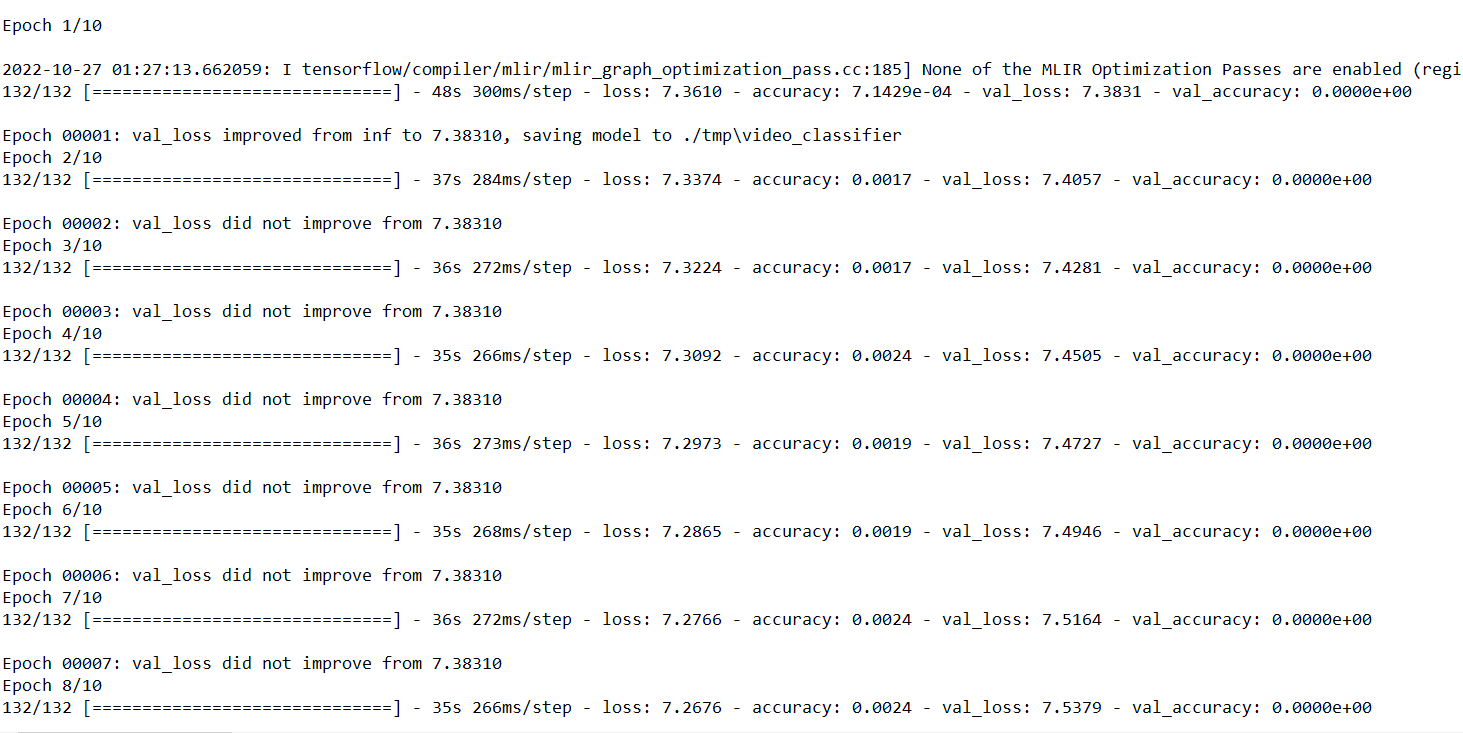
\_, accuracy = seq\_model.evaluate([test\_data[0], test\_data[1]], test\_labels)

print(f"Test accuracy: {round(accuracy \* 100, 2)}%")

return history, seq\_model

\_, sequence\_model = run\_experiment()

s1 = [t0, t1, ... t100]  
s2 = [t101, ... t201]  
...  
s16 = [t1501, ... t1547]



**7.4 Precompute CNN feature map**

As mentioned above a CNN generates the rules or data patterns by taking data and results.

This data has to be prepared

The training process uses the data patterns or rules that have been extracted and gives the trained weights. Based on the learned data patterns, comes the prediction, which provides the class of the frames.

**7.4.1 Prepare Single Video**

Before starting the training, process must prepared its inputs. Using nympy.zeros which output is a new array of given shape and type, filled with zeros. For the videos is used

frames = frames[None, ...]

frame\_mask = np.zeros(shape=(1, MAX\_SEQ\_LENGTH,), dtype="bool")

frame\_features = np.zeros(shape=(1, MAX\_SEQ\_LENGTH, NUM\_FEATURES), dtype="float32")

Also, the videos has to be shorter for the pad, doing the follow.

for i, batch in enumerate(frames):

video\_length = batch.shape[0]

length = min(MAX\_SEQ\_LENGTH, video\_length)

for j in range(length):

frame\_features[i, j, :] = feature\_extractor.predict(batch[None, j, :])

frame\_mask[i, :length] = 1 # 1 = not masked, 0 = masked

return frame\_features, frame\_mask

## **7.5 Model training and inference**

Feature maps are generated by applying Filters or Feature detectors to the input frames or the feature map output of the prior layers. Feature map visualization will provide insight into the internal representations for specific input for each of the Convolutional layers in the model.

<https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c>

This project extracts features from the frames of each video, as the code explains bellow.

for i, batch in enumerate(frames):

video\_length = batch.shape[0]

length = min(MAX\_SEQ\_LENGTH, video\_length)

for j in range(length):

if np.mean(batch[j, :]) > 0.0:

frame\_features[i, j, :] = feature\_extractor.predict(batch[None, j, :])

else:

frame\_features[i, j, :] = 0.0

return frame\_features

**7.5.1 Predict Action**

In the end, the output of the CNN is the predict action. The labeling process with StringLookup converted to vocabulary form, for a proper print. As the frames and the input of the feauture map , on which the training applied. The code is below.

def sequence\_prediction(path):

class\_vocab = label\_processor.get\_vocabulary()

frames = load\_video(os.path.join("test", path))

frame\_features, frame\_mask = prepare\_single\_video(frames)

probabilities = sequence\_model.predict([frame\_features, frame\_mask])[0]

for i in np.argsort(probabilities)[::-1]:

print(f" {class\_vocab[i]}: {probabilities[i] \* 100:5.2f}%")

return frames

Test video

In the end , the final output

test\_video = np.random.choice(test\_df["video\_name"].values.tolist())

print(f"Test video path: {test\_video}")

test\_frames = sequence\_prediction(test\_video)

**8. Experimental Results**

**9. Discussion**

1. **Conclusions and future work**

**Technical part**

When there are variations in between the frames of a video not all the frames might be equally important to decide its category. In those situations, putting a [self-attention layer](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Attention) in the sequence model will likely yield better results.

3.3 The Transformer Architecture

The importance of focus in some particular information is the goal of this Transformer Architecture. To succeed that the network must decide which regions is more important than other and using for mining the useful data from them and discard-or fade out- the rest.

For this scope, the main process is to compute the scores-higher or lower- depending on the relevance of each feature.

<https://aws.amazon.com/getting-started/guides/deploy-webapp-elb/module-three/>

<https://www.researchgate.net/figure/CNN-RNN-architecture_fig1_353254139>

<https://arxiv.org/ftp/arxiv/papers/1604/1604.04573.pdf>

First, self-attention layers that form the basic blocks of a Transformer are order-agnostic. Since videos are ordered sequences of frames, we need our Transformer model to take into account order information. We do this via **positional encoding**. We simply embed the positions of the frames present inside videos with an [Embedding layer](https://keras.io/api/layers/core_layers/embedding).

<https://keras.io/examples/vision/video_transformers/>

3.1 Positional Encoding Layer

Positional encoding describes the location or position of an entity in a sequence so that each position is assigned a unique representation. There are many reasons why a single number such as the index value is not used to represent an item’s position in transformer models.

Transformers use a smart positional encoding scheme, where each position/index is mapped to a vector. Hence, the output of the positional encoding layer is a matrix, where each row of the matrix represents an encoded object of the sequence summed with its positional information.

3.1.1 Embedding Layer

# This layer can only be used on positive integer inputs of a fixed range. As will be mentioned in the chapter of preprocessing Dataset, to do that a StringLookup layer will be used.

**The Embedding Layer accepts, in this network, tf.Tensor which has a single data type and a shape. The model that has been used has an** integer matrix of size ,and the largest index

<https://keras.io/api/layers/core_layers/embedding/>

A technique that’s called Positional Embedding adds a position embedding to the corresponding word embeddings to obtain a position-awatr word embedding

(photo 27)

3.2 Transformer Encoder

Factoring outputs into multiple independent spaces , adding residual connections, adding normalization layers-all of these are standard architecture patterns that one would be wise to leverage in any complex model. Together, these bells and whistles form the Tranformer encoder.(photo tr14) (Subclassed Layer) .

For sequence data a good idea is to use LayerNormalization layer, which normalizes each sequence independently from other sequences in the batch(photo tr22). This layer pools data within each sequence separately with the scope to use it to assemble an image-classification model.

A transformer is a sequence-to-sequence model. That means that it was designed to convert one sequence into another.

Diagram, schematic

Description automatically generated

##### The TransformerEncoder chains a MultiHeadAttention layer with a dense projection and adds normalization as well as residual connections.

<https://livebook.manning.com/book/deep-learning-with-python-second-edition/chapter-11/274>

3.2.1 Self Attention Layer

These Layers build a self-attention mechanism to create global dependencies between inputs and outputs of the layer inside the Transformer-based Architecture.

The purpose of self-attention is to modulate the representation of a token by using the representations of related tokens in the sequence. The frames of video is used to clarify what kind of the human move is, depending on the others frames of the sequence.

(tr3)

The self-attention mechanism allows the inputs to interact with each other (“self”) and find out who they should pay more attention to (“attention”). The outputs are aggregates of these interactions and attention scores.

## 1. Illustrations

The illustrations are divided into the following steps:

1. Prepare inputs
2. Initialise weights
3. Derive **key**, **query** and **value**
4. Calculate attention scores for Input 1
5. Calculate softmax
6. Multiply scores with **values**
7. Sum **weighted** **values** to get Output 1
8. Repeat steps 4–7 for Input 2 & Input 3

<https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a>

The MultiHeadAttention layer that Keras provide as an built-in layer is the way to handle a vectorized implementation

+phototr25

3.4 Utility Function

The ability that the human brain has to rank a state of an image to the word, can be achieved with a Utility Function into the neural network. This function takes a state and returns a value that this state represents.

In a video showing a person to present a word in sign language, a utility function learn to rank states by extracting the first state of the heads and the last state as the training data.

In this case the Tranformer encoder which show before combined with the PositionEmbedding layer ,as a normal Embedding layer. To take word order into account must swap the old Embedding layer with the position-aware version.

A picture containing shape

Description automatically generated

<https://livebook.manning.com/concept/machine-learning/utility-function>

This is a good practice to rank the value of a word order information for classification. The code of these process presents on the next image.

Simeiosologia oti einai entaksei

Sto train tis 3 k sto test tin 1

1x1576

4000 video me 1000 lekseis

K na dokimasw diaforetika megethi leksikou k sto telos to 1000

Ana leksi

**11.Bibliography**