UNIVERSITY OF RZESZÓW
Faculty of Exact and Technical Sciences

Jakub Flis

Student ID number: 117794

Computer Science

Camera Movement Classification Using a Deep Learning Model Trained on Synthetic Video Clips

Master's Thesis

Supervisor:

Michał Kępski, PhD

Rzeszów, 2025

Table of contents

Introduction

Deep Learning (DL) is a rapidly growing field that has been followed closely by the public eye ever since the Transformer architecture was used in Large Language Model research. Hardware improvements and the recent refinement of deep architectures has put much attention on the data acquisition process. For the field to evolve further, increasing amounts of high-quality data are required. Real-life data can only be obtained at large scale by big enterprises and due to copyright and privacy concerns its public sharing is limited. For this reason, alternatives like synthetic data generation augmented by procedural content generation have emerged in the past decade to facilitate further Machine Learning (ML) research.

One example of a field that requires alternative, freely available data sources is cinema. While analyzing cinematic movies is not a domain famous for the need of deep learning insights, studying the dynamic camera movements used to: control movie scenes, captivate the audience and to reflect its budget could be beneficial to understand the success of some shows. This is the very essence of a movie analysis field known as cinemetrics[17].

Mirroring the sense of sight, camera is the most important sensor type that is not only an important force driving the entertainment industry, but can be used to extract invaluable information (e.g. depth field, optical flow) about the world by machines. At the intersection of DL fields like: camera understanding, depth field analysis and 3D model understanding, with future advances new groundbreaking applications could be found like cheaply modeling a virtual 3D scene based on a video of a scenery made by a smartphone.

Computer vision is a field that is closely connected to visual information. It utilizes algorithms and deep learning to process sensor data in a similar way that humans do[41]. Due to advancements in automation, approaches connecting different sources of data, as well as new methods of extracting visual information have been researched at scale. This is helpful in applications like: handwritten text recognition, quality control on assembly lines in factories, video tagging, medical diagnostics and the general task automation.

Analyzing images is a complex task, which is caused by the fact that an image is a very imperfect way of capturing reality. Information like depth of field is not represented in a clear

way and each object can have an infinite number of two-dimensional shape representations depending on its pose and distance to camera. To depict an object more closely it could be very valuable to record a video of it but then the task of finding the outline of the specific object and combining the sequential, spatiotemporal information about it from every frame is an even more complicated undertaking, which then also needs to take into account anomalies in shape and the movement of the sensor itself (Figure 1).

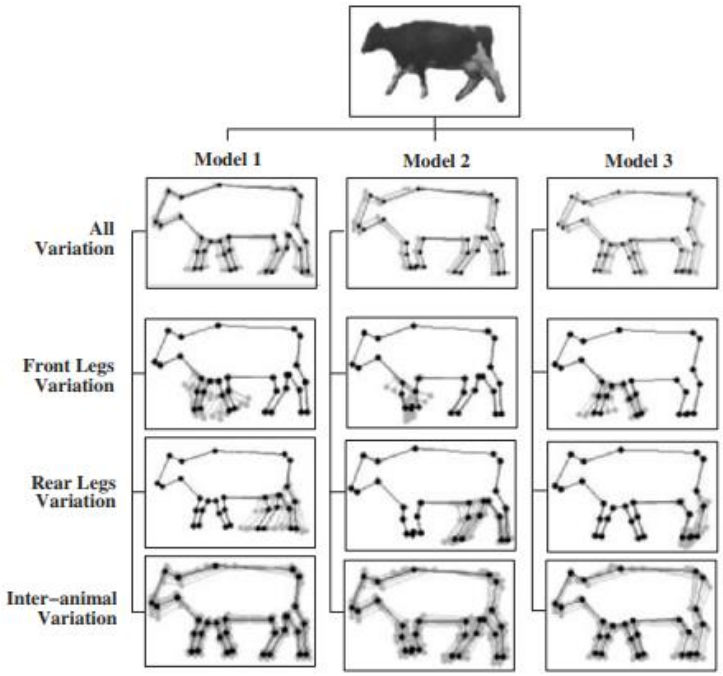

Figure 1. Defining the 'cow' class with its variations. Reprinted from [35].

Synthetic data generation opens up many possibilities but the approach is not free of limitations. Data created using an imperfect model of reality is prone to generalization issues, biases and a general lack of variety. Synthetic generators need to be able to provide enough high-quality, varied data to enable training of modern architectures with high parameter counts. Procedural content generation is a field that can provide that by allowing to quickly create variations of a scene using flexible algorithms controlled by a set of easily-modifiable parameters[30].

The thesis aims to utilize procedural content generation in order to create a video dataset that can be used for data augmentation in dynamic camera shot classification. For this purpose, a clip generator application will be created and presented in this document. To ensure high quality of the generated data, the existing synthetic-data research will be reviewed and used to propose a framework for generating synthetic dynamic video shots. To

determine the exact parameters of the generated dataset, established datasets will be studied. Finally, the generated video dataset will be tested by training DL models based on well-established architectures from the field of Human Action Recognition (HAR).

The hypotheses investigated in this document state that the procedural approach can be effectively applied to the task of dynamic camera shot classification, and that the quality of the generated data can be evaluated by testing the trained models on unseen environments.

The thesis is split into 3 chapters. Chapter 1 discusses the dynamic camera shot classification problem, overviews existing research, describes relevant video datasets and attempts to make a connection between movie shot classification and HAR tasks. Chapter 2 provides a deeper theoretical understanding of modern DL methods. Chapter 3 is dedicated to presenting the results including the created synthetic video generator and the prototype of the synthetic dynamic shot dataset along with tests performed on popular DL architectures (C3D and Timesformer) for videos.

1. Definition of the problem

1.1. Characteristics of the problem

The considered problem of camera movement analysis was reduced to a classification problem where camera movements are assigned to 1 of the 15 classes described further in the chapter 3. Classification is one of the two main objectives of machine learning, along with regression, and it involves making a decision and assigning a label to an instance – therefore assigning it to a class. A class is a term describing objects with defined common traits. Classes can either be defined manually or automatically by algorithms[41]. Due to the complexity of the problem and the number of defined classes, an algorithmic approach was rejected in favor of deep learning methods. These methods require large amounts of high-quality data. This can be overcome by using an increasingly popular approach of synthetic data generation[30].

The main benefit of synthetic video generation involves the ability to generate endless data, while also being able to make adjustments to the generator in response to the training results. This allows to increase the data quality to the required level. The primary goals when designing a synthetic video data generator for classification tasks is to achieve the maximal visual fidelity to the real-world data. Each real-world class needs to be closely studied in order to recreate it synthetically[8].

One of the major issues in synthetic video data generation is the fact that it introduces many new data domains to each task. Being able to combine related datasets when data is scarce is always beneficial but is problematic due to differences between data sources and interpretations of a problem (e.g. class definitions, data quality)[15]. In the considered problem of camera shot classification there is a pronounced lack of labeled real-world datasets due to copyright issues. Existing or related datasets are very difficult to be combined because of the class definition problem and because of the overwhelming overrepresentation of static shots in popular video datasets like Kinetics-400.

Real-life cinematic scenes can be very complex with intricate backgrounds and crowds of character moving in all directions. Generating synthetic data capable of replicating these qualities is needed for training generalized models, which is also equivalent with the data being high-quality. It is important to include scenarios where[1]:

- Object occlusion happens – the trained model should not rely on following a single target. Detailed sceneries with much clutter must be included, which will inevitably lead to objects restricting vision. The complication will help the model generalize to unseen data.
- Uncommon camera angles are utilized – one of the most potent tools at movie director's disposal is surprise. Equipping the synthetic data generator with unusual parameter values could help the trained model generalize better.
- Multiple objects move in different directions – the generator should possess a pedestrian managing system, with character moving and performing typical actions. Car is also another example of a moving object that is common in movies and could be included to improve data variety.

1.2. Limitations of using real-life cinema video datasets

Creating a video dataset is a big undertaking requiring lots of man-hours and addressing many difficulties. Video is a modality characterized by requirement of large storage. It can be considered much less efficient than images in conveying the same amount of information, as motion is more difficult to represent and involves repeating most of the information every frame (high information redundancy). Creating real-life video generally takes more effort than taking photos, and browsing videos also takes significantly more time when labeling is required.

The most common sources of video information consist of cinema movies, video platforms like YouTube and video recordings from security cameras. For this thesis cinema movies would provide the most accurate source of training data. However, most of this data is not only protected by copyright but also not freely available. To circumvent the issue, promotional footage like movie trailers can be used for personal, academic use. Trailer YouTube links can be retrieved through a movie website (e.g. The Movie Database offers a publicly available API that allows to automate trailer links retrieval). Due to time constraints of the thesis this source of data was not pursued. Legal limitations still need to be considered.

One source of public-domain, real-life cinematic movies worth investigating is Prelinger Archives – an archive project with the goal of preserving culturally significant movies[27]. The database currently contains 4,928 downloadable English movies of varying length, released in

years 1903 – 2025. 65% of those movies belong to the public domain and would not pose any copyright problems. The movies tend to be niche and low-budget and they might not involve the level of camerawork that would suit the reality modelled in this thesis. Since the database is a subset of Internet Archive, API is available to obtain the list of items and also individual download links can be retrieved automatically, which can help speed-up the data acquisition process.

YouTube platform is a source of data used in datasets like Kinematic-400 but this dataset not suitable for this task as little camera work is used there – the vast majority of YouTube videos consist of only static shots. Not only that, but due to law regulations, datasets like Kinematic-400 are regularly maintained and decay gradually because of creators retrieving their permission for their videos to be included[1].

Regardless of the chosen data source, preparing a real-life video dataset for a niche classification problem with 15 classes requires manual labeling of at least one thousand clips but that might require processing many more videos in order to find an adequate number of clips belonging to rare classes. It is extremely difficult to provide a rare class like dolly zoom with enough real-life instances.

The most significant issue behind creating a cinema video dataset comes from the task of splitting movies into scene clips. The task is known as ‘shot boundary detection’ in computer vision and it is still not a solved problem, with state of the art DL approaches achieving an averaged F1 score of 0.92 in 2020[34]. This means that the process of clip cutting cannot be fully automated and will take a considerable amount of time, which further emphasizes the need for synthetic solutions.

1.3. Overview of movie classification and synthetic data generation

The task of classifying clips based on objects motion over time is closely aligned with human action recognition subfield of computer vision, since both applications are unique in the way that they require multi-frame information with a method for analyzing both spatial and temporal relationships simultaneously. Methods that can be used for classifying camera motions are very similar as methods used in human action recognition.

CineScale[31] is a multinational academic project with the goal of publishing a cinema-movie clips dataset and preparing a simple CNN model for the task of automated shot scale recognition. The dataset consists of 792,000 image frames from 124 full movies, each extracted at 1 frame per second across 9 classes as shown on Figure 1.1. This application differs significantly from dynamic camera shot classification because the data was used as separate images and not actual videos. It might be possible to combine the frames into videos but it does not promise success with the very low frame rate. Another thing of note is the approach to copyright taken in the article. Complete, copyrighted movies were used as a data source and the acquired data was published in JPEG format with each image being one frame of a movie. The only action taken by the authors to prevent copyright infringement was using modified movie frames as class examples in the article as shown on Figure 1.1.

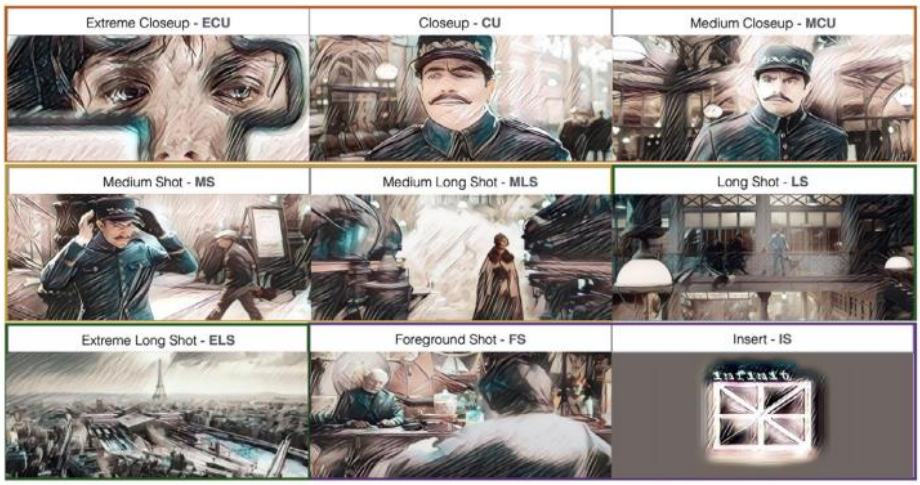

Figure 1.1. Shot scale classes used in Cinescale. Reprinted from [31].

In 2022[24][25] a movie shot classification feature dataset was published. It consists of 10 dynamic shot classes that differ from the ones used in this thesis – notably: handheld, aerial and vertical-tilt shots were included, and the omitted were: zoom-out, arc and roll. The dataset consists of 1803 video clips – 55% of them are static shots, 15% are handheld and 11% are pan shots. The extreme unbalance shows the issue with using real-life data for dynamic shot classification. It also shows the importance of handheld shot types which have not been considered further in the thesis. The original movie data was not published due to copyright.

A unique view on cinema shots was proposed in 2020 in paper titled ‘A Unified Framework for Shot Type Classification Based on Subject Centric Lens’[29]. The paper proposes a dual-stream architecture designed specifically for simultaneous (but logically

separate) scale and movement (Figure 1.2) classification. The paper points out the importance of the background in determining dynamic movement type as well as the shortage of publicly available movie datasets designed for movement classification. The project adopted only 4 movement types, which seem to decently represent the vast majority of practically used shots in cinema. The authors compiled a video dataset (unpublished due to copyright) comprising 46*K* clips with scale and movement labels, using 7*K* publicly available movie trailers. Additionally, in order to maintain quality of clips and labels, every step of the process involving removing advertisement information and clip splitting was first performed using state-of-the-art computer vision methods and then verified manually by cinematography professionals. While it is of little relevance to the thesis, it is worth noting that the authors combined shot scale classification together with subject map generation (determining the most important object in the frame) in the application of automated shot editing – generating frame variations in other possible shot scales, which shows further that camera analysis is potentially worthwhile.

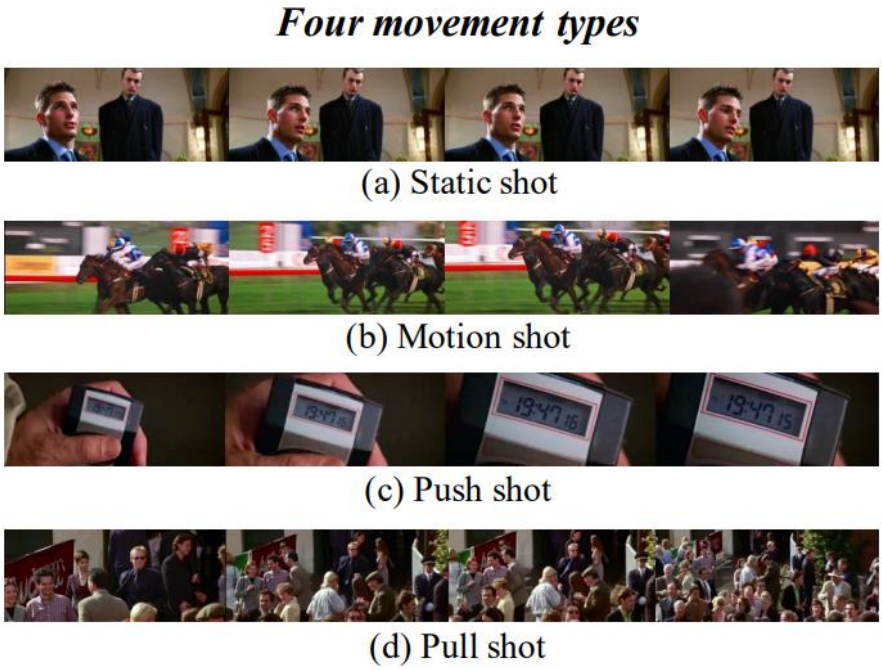

Figure 1.2. Camera movement classes defined in MovieShots dataset. Reprinted from [29].

CARLA[13] is a well-known in autonomous driving research, open-source simulator created in Unreal Engine 4. It aims to recreate realistic car driving in urban environments. It was designed to work together with a programmable Python API which allows the user to access and modify every aspect of the scene at any given moment. The simulator features

multiple, decently-sized maps with significant background variations. The realism was established in the following ways:

- Realistic lighting and weather conditions (Figure 1.3) – pixel brightness and image color distribution is an important aspect in most computer vision methods. The simulator enables controlling day and night cycle, rain and clouds, all of which posing a challenge in computer vision applications.
- Varied car and pedestrian models – variety is necessary for models trained on RGB input for object recognition. If one graphical model was used for each obstacle class, the trained model would struggle to recognize a class when faced with any different graphical model and it would worsen the generalization of learned patterns into real-life data. Pedestrians are generated with pre-specified wardrobe and might be walking distracted by using a smartphone.
- Varied urban sceneries – the simulator is equipped with more than 10 distinct maps – this is especially important in autonomous driving as each map comes with new road layouts, forcing trained models to generalize to new scenarios by deepening their understanding of realistically implemented traffic rules. The maps are also populated by pedestrians who might walk onto the street unexpectedly.
- Realistic driving physics – the cars have realistic braking times and autonomous driving models are able to learn how early and how strongly they need to brake in any scenario.

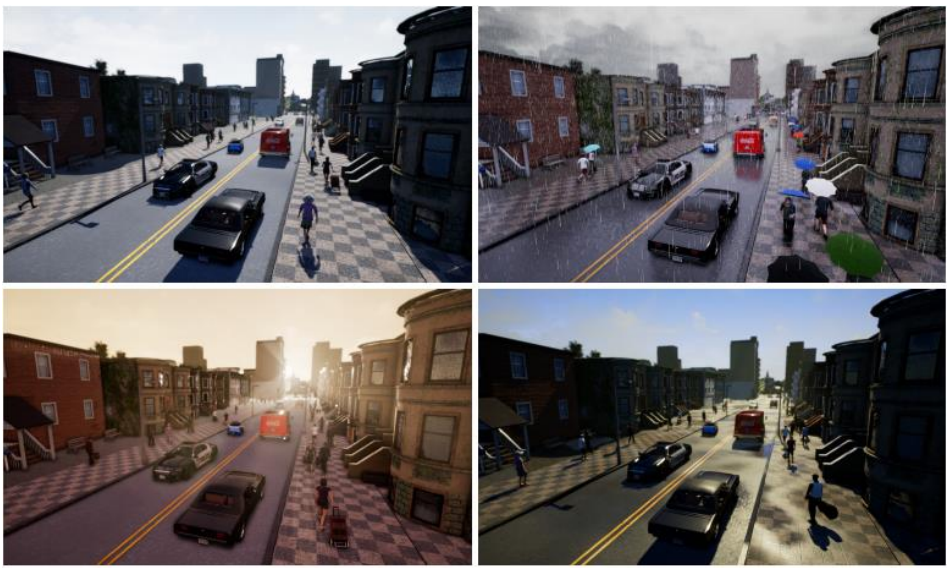

Figure 1.3. Weather and lighting conditions in CARLA. Reprinted from [13].

CARLA API can be used as a solid foundation for other uses from the intended. Specifically, it could be used as another source of synthetic data for dynamic camera shot classification. The simulator's API provides every tool needed:

- Camera can be controlled by a Python script running at every frame.
- Each building's position and its bounding box can be accessed[5] and used to precompute a texture encoding information about correct camera positions so that the camera will not collide with the environment.
- Pedestrians and cars can be freely controlled and spawned.
- Simulation time-step can be adjusted to generate as many frames per second as the server can handle.

Data generated in this manner would have more than adequate graphical fidelity in order to generalize to real-life video. Including CARLA-generated data into a training dataset for camera shot classification is a recommended and relatively undemanding way to increase the variety and quality of a multi-domain synthetic dataset. Such dataset should allow to decrease the volume of required real-life data while keeping the model accuracy at desirable levels. Two major downsides of working with CARLA are its relatively high hardware requirements (the GTX 1060 3GB does not have enough VRAM to launch it) and its lack of scenery variety – the simulated environments are urban and do not include other common filming locations like closed spaces, nature views or uncommon architectural styles.

In 2024 another synthetic video generator in Unity was published[8], designed to augment a human action recognition dataset with 11 classes. The purpose of the project was to automate detecting sports exercises in order to keep users accountable to their fitness goals. The generator worked by augmenting real-life videos of humans exercising (Figure 1.4). The augmentation was performed first by using pose landmark detection from *MediaPipe* library on real-life videos, and then by applying digitalized animation to fictional, procedurally-generated avatars created using *SyntheticHumans* package in Unity. Using the synthetic clips proved to be an almost-perfect replacement for a scarce real-life dataset, and it also worked fairly well as an augmentation technique. These results were obtained by increasing the scene variety by randomizing: avatar model, background, camera position and lighting. Authors used

real-life, previously unseen data for testing the generalization ability of the synthetic data by utilizing I3D and Timesformer architectures.

Figure 1.4. An example of a synthetic data generator made with Unity. Reprinted from [8].

1.4. Analysis of real-life video HAR datasets

Creating a real-life video dataset is an immense undertaking, which comes from the fact that state-of-the-art architectures for video tasks require enormous volume of training data to converge. Not only that, but capturing enough instances of rare classes from real-life data is very difficult. The cost compounds further as manual assignment of labels is very expensive and can still be inaccurate. For this reason big, real-life computer vision datasets are scarce and are typically provided by large enterprises with the required resources.

Datasets have a multitude of uses in deep learning, as they can serve both as a training foundation, as well as a benchmark for existing models. In the early years of computer vision, datasets used to be considered a challenge to be solved by itself and cross-validation enabled to verify generalization ability of models[19]. As architectures developed and started to become quite large, singular datasets no longer are enough to verify quality of models. State-of-the-art models possess too many parameters to be trained using a single dataset (with the exception of the largest datasets like Kinetics), which is why transfer learning is necessary to use the best architectures with most of the HAR datasets. For this reason, data scarcity is the biggest problem in HAR[1].

The first two major real-life video datasets commonly used in HAR were HMDB-51 and UCF-101. HMDB-51 (Figure 1.5) was published in 2011 and consists of 51 classes with each containing more than 100 unique clips summing up to 6,766 video clips total. To ensure label

correctness each clip was verified by two observers. The dataset is versatile thanks to detailed meta tags describing: camera position, camera motion, video resolution and the number of people in the scene. The data was collected from varied sources like: YouTube, Google, Prelinger Archives and others. All video frames were normalized by resizing them to 240p resolution with original aspect ratio intact. To reduce the required disk space, *DivX 5.0* codec compression and *ffmpeg* video library were used. As an additional pre-processing step, video stabilization on handheld shots was performed[19].

UCF-101 (Figure 1.5) was introduced in 2012, doubling the number of actions and the total number of video clips compared to HMDB-51, bringing them up to 101 and 13320 respectively. The authors used YouTube as the data source, downloading the clips in 25 FPS and 320 x 240 resolution. The chosen file format was *.avi* and compression was done using *DivX* codec from k-lite package[36].

Figure 1.5. Example frames from HMDB-51 and UCF-101. Reprinted from [19][36].

As visible in Table 1.1, driven by the development of deep learning architectures, new and bigger datasets kept being created. To test the scalability of architectures, the number of action classes kept increasing, as well as their total volume. The primary sources of video data for HAR always were cinema movies and YouTube but in the past decade synthetic simulators with procedural content generation have been utilized in more specific HAR applications because they are able to retrieve more modalities of data from a scene (e.g. depth field, exact motion information)[30]. However, hired actors can also provide a commendable source of data. The EPIC-KITCHENS-100 dataset was published in 2021 and was created by equipping 21 participants with a GoPro head camera and instructing them to vocally describe each kitchen

action they were taking. This resulted in a large, multi-modal dataset with footage adding up to respectable 100 hours[11].

Dataset	Number of Actions	Clips	Background	Camera Motion	Release Year	Resource
KTH [11]	6	600	Static	Slight	2004	Actor Staged
Weizmann [3]	9	81	Static	No	2005	Actor Staged
UCF Sports [10]	9	182	Dynamic	Yes	2009	TV, Movies
IXMAS [12]	11	165	Static	No	2006	Actor Staged
UCF11 [6]	11	1168	Dynamic	Yes	2009	YouTube
HOHA [7]	12	2517	Dynamic	Yes	2009	Movies
Olympic [8]	16	800	Dynamic	Yes	2010	YouTube
UCF50 [9]	50	6681	Dynamic	Yes	2010	YouTube
HMDB51 [5]	51	6766	Dynamic	Yes	2011	Movies, YouTube, Web
UCF101	**101**	**13320**	Dynamic	Yes	**2012**	YouTube

Table 1.1. Comparison of early video datasets. Reprinted from [36].

The most well-known dataset in the entirety of video deep learning is Kinetics. It was published in more than one version, but the most popular is Kinetics-400 consisting of 400 classes with more than 400 clips per class and each clip lasting 10s on average. The dataset is a spiritual successor to HMDB-51 and UCF-101, exceeding UCF-101 clip count by a factor of 23. Due to its sheer size and varying class rarity, the dataset is moderately unbalanced. The labeling process was automated but required building a graphical interface to perform manual label-review process (Figure 1.6). The most relevant point to this thesis, that was raised by the authors, concerns 'noisy' classes. Some of the class distinctions are less pronounced from the others, leading the model to struggle with their distinguishing. To some extent this noise might be a positive occurrence that can make trained models more robust, but sometimes the reviewers themselves struggled to tell class instances apart. The way the problem was handled by the authors of the dataset was by manually reviewing problematic classes (Table 1.2) and by merging, splitting or removing them to prevent ambiguity[18]. The dataset still remains the most important dataset for video architectures and is commonly used to pretrain parameter-intensive models before performing transfer learning to smaller, more specialized datasets.

Many more real-life video datasets have been published but due to their highly-specialized nature they were deemed less relevant to the thesis. Many of the established video datasets contain additional metadata and pre-arranged splits, with manually defined: training, validation and test subsets. Metadata appears to be of significant help when attempting to use a dataset outside of its intended applications. Videos and each scene can tagged with labels for: visible objects, speech transcriptions, scene type, location and other data that might be registered with additional sensors. Considering the advanced stage of modern video deep

learning architectures, the future of HAR might strongly involve the creation of multi-modal datasets[1][40].

Figure 1.6. Mechanical Turk labeling interface used for Kinetics-400. Reprinted from [18].

Class 1	Class 2	confusion
'riding mule'	'riding or walking with horse'	40%
'hockey stop'	'ice skating'	36%
'swing dancing'	'salsa dancing'	36%
'strumming guitar'	'playing guitar'	35%
'shooting basketball'	'playing basketball'	32%
'cooking sausages'	'cooking chicken'	29%
'sweeping floor'	'mopping floor'	27%
'triple jump'	'long jump'	26%
'doing aerobics'	'zumba'	26%
'petting animal (not cat)'	'feeding goats'	25%
'shaving legs'	'waxing legs'	25%
'snowboarding'	'skiing (not slalom or crosscountry)'	22%

Table 1.2. The most common class confusions in Kinetics-400. Reprinted from [18].

2. Theoretical background

2.1. Understanding motion using optical flow

Motion is a relative change in position of objects in a scene. The relativity becomes a significant challenge in computer vision tasks when the camera and some objects change their positions simultaneously. Movement of the camera causes all objects in the scene to move in the opposite direction with perceived speed relative to the distance between each object and the camera. (depth of field could be used to represent the motion better). Fortunately, modern DL-based methods are already capable of decently inferring those complex relationships based on RGB video alone[32].

Motion representation achieved using optical flow still remains useful. Optical flow is a 2D vector field describing perceived translations of brightness within image neighborhood. Pixels with similar brightness often belong to the same object and this is used as an approximation of edge detection during object motion. This assumption requires consistent brightness between subsequent frames of a scene. For best results videos with high frame rate should be used as low frame rate makes fast objects impossible to track effectively[41].

Optical flow can be approximated using numerous methods and most methods are capable of outputting a dense flow field that assigns a 'movement' vector to every single pixel (as opposed to a sparse vector grid) which then can be used as an additional channel of image representation in neural network input. When choosing a technique for evaluating optical flow several limitations need to be considered[1]:

- Motion speed and frame rate – if object positions change significantly with each frame then methods that take larger neighborhoods into account become necessary but require notably more resources.
- Brightness consistency – most methods become inaccurate with scene illumination changes. Methods for overcoming this limitation were proposed as recently as 2023[22].
- Image quality and potential noise.

- Computational resources – robust, accurate methods like TV-L1 require high compute times. More balanced approximation methods like Farnebäck are popularly used otherwise.

In the context of the future research related to this thesis, optical flow could provide extensive additional information about a scene but would require the generated datasets to consider additional requirements, like increasing the frame rate for calculating accurate optical flow for fast camera movements and handling the increased dataset size from storing the flow representations.

Optical flow is very expensive computationally to calculate using classical algorithms. For this reason, flow models like PWC-Net (2017)[39] or distillation approaches like D3D (2018)[37] are used in order to approximate the flow at cheaper cost. Modern deep learning methods of flow estimation achieve state-of-the-art accuracy compared to classical methods. The problem of evaluating flow-accuracy points to the crucial role of synthetic data in DL – it is unrealistic to gather ground truth optical flow labels from real-life video. This is commonly done using digital environments like 3D animation and game engines that allow to retrieve positions of each object in the scene with very high accuracy and for this reason datasets like KITTI 2015 (compiled using autonomous driving simulator made in Unity)[4] and MPI Sintel (made from a short 3D animated movie) were used as ground truth in the early days of flow models.

Optical flow allows to focus on scene changes on a frame-to-frame basis, without distracting the model with unneeded information like object appearance. Optical flow could be considered a method of data augmentation, increasing the amount of information that can be observed in video. 2017 paper ‘On the Integration of Optical Flow and Action Recognition’ by Sevilla-Lara et al. proposes that optical flow estimation models should be trained with loss function calculated based on not the ground truth optical flow data but on the classification error of the end-goal task[32]. The findings imply that unrealistic, domain-specific optical flow could be superior to ground truth.

2.2. Convolutional neural networks, training and hyperparameters

Machine learning is a very data-driven field, consisting of numerous methods and tools that get chosen specifically for a given task. It is theoretically possible to perform classification tasks on images using the vanilla fully connected deep neural networks, however it is a naive approach that involves prohibitive computing costs. Its inefficiency can be demonstrated using an image sized 224 x 224 x 3. Assuming the hidden layer contains 1000 neurons, the parameter count would exceed 224 x 224 x 3 x 1000 = 150 million. For comparison, the GPT-3 model contains 175 billion parameters which is a number that is far too close considering the complexity difference. However, the computational cost is not the only shortcoming of the naive approach. Fully connected networks do not naturally possess any mechanism for capturing the spatial relationships between each individual pixel. This means that applying any spatial transformation (e.g. rotations, translations) to the input image would greatly worsen the end results.

The first task in history that called for a new approach to graphical data was handwritten digit classification (Figure 2.1)[20]. The article was one of the first ones to adapt the backpropagation method to work for kernels of convolutional neural networks (CNNs), achieving both high accuracy and low training times.

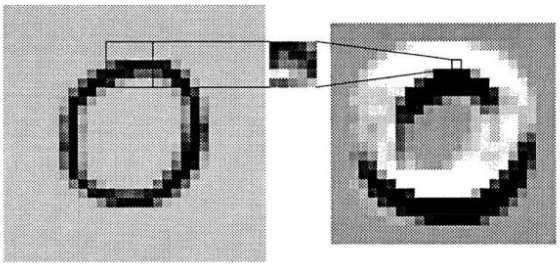

Figure 2.1. An example of a convolution with a kernel and the resulting activation map. Reprinted from [20].

The phenomenon explaining the relative advantage of CNNs is known as sparse connectivity. Traditional neural networks require matrix multiplication for every possible pair of neurons between the input and output layers. Assuming *m* inputs and *n* outputs, this leads to the computational complexity of *O(m x n).* A convolutional kernel sized *k* only calls for *O(k x n)* runtime, which is orders of magnitude faster, since kernels are typically very small. This leads to the so called parameter sharing. Instead of updating weights for each and every connection between neurons, only the kernel weights need to be updated and each of them

will be applied multiple times – approximately equal to the number of inputs. While parameter sharing does not improve the forward pass complexity of *O(k x n)*, it greatly reduces the number of parameters[14].

Convolution is a mathematical linear operation that works using a custom-defined kernel on input data that can be arranged in a grid. Examples of such data include: images, time series and sounds. The convolution operation replaces traditional matrix multiplication operations in neural networks thus significantly increasing performance and allowing the model to understand spatial relationships[41].

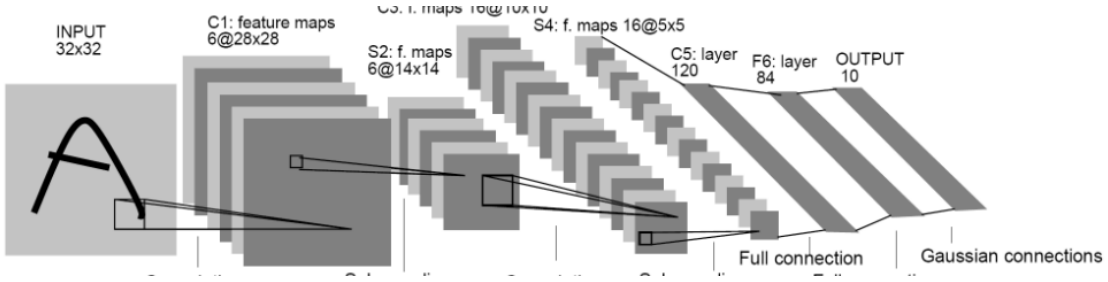

Figure 2.2. An example of a CNN architecture. Reprinted from [41].

The actual term for the 'convolution' operation used in convolutional neural networks is cross-correlation. The operation is very similar to convolution but it does not involve rotating the kernel matrix. From a mathematician's standpoint the convolution has more valuable properties, however they are not relevant to computer vision applications. For this reason it is customary to refer to cross-correlation as convolution inside the computer vision field[14].

Convolution is a mechanism allowing CNNs to understand the spatial relationships of an image (Figure 2.2). Each convolutional layer of a network uses an arbitrary number of kernels (also called filters) with each one capable of detecting one of more visual patterns called features. The output of a convolution of the input and the kernel is called a feature map and it contains the information about where the kernel-specific features occur in the image and how pronounced they are. Convolution affects the size of the input based on the following 4 hyperparameters[41]:

- Stride - controls the way the kernel traverses the image pixels. When set to 1 while using padding the image dimensions will not change. Stride set to 2 allows to downscale the input x and y dimensions by a factor of approximately 2.

- Kernel size and convolution type - a convolution can either be valid or full. A valid convolution is only performed for a given pixel if the kernel does not exceed the boundaries of an image. This means that a convolution with a kernel size of 3 will not be performed for pixels on the border of an image, which will reduce both the width and height of the output feature map by 1. This is the default behavior for a convolutional neural network. The full convolution works in the opposite way, being performed for every pixel of an image returning a feature map bigger than the input.
- Padding - extends the output borders using zeros increasing its size. Typically convolution reduces the input size, requiring padding in order to either preserve the original input size (in case of stride set to 1) or to keep the dimensions as even numbers (in case of stride set to 2).

As each kernel is a set of weights, the linear matrix multiplication causes the output of the convolution to be a linear activation in the same way as in traditional neural networks, After the convolution, during the detector stage, a nonlinear activation function like the rectifier (*ReLU*) is added[14].

The output of a single convolution is a feature map with the depth of 1. The output of a convolutional layer is a stack of all calculated feature maps. Output's depth channel's size is equal to the number of filters used in the convolution. The stack represents all the patterns recognized by each filter. The effectiveness of the pattern recognition is entirely dependent on assigning proper weight to the filters. Through the process called backpropagation a convolutional network adjusts the filter weights in order to recognize as varied and impactful patterns as possible[41].

After obtaining the stack of filter maps, the pooling stage is administered. The first convolutional layers in every architecture tend to have rapidly increasing depth dimensions of the output. To make the model computationally feasible, the down-sampling property of the pooling becomes valuable. Most of the layers in convolutional networks work either by increasing dimensions of the output to facilitate feature extraction, or by decreasing its dimensions to reduce computational overhead and to aid the model in combining lower-level features into more abstract patterns. Pooling is a function that returns one representative value for each rectangular neighborhood of an image. The most commonly used variant of pooling, max pooling, can be described as an operation that divides the input into equally sized

regions and replaces each one of them with the highest value from inside of each region. Pooling is the process that also introduces translation invariance to convolutional networks. By the nature of convolution, a pattern can only be detected when the exact pattern can be seen on an image. Which means that any change of scale or rotation would prevent any pattern from being detected. Max pooling causes loss of information by removing the majority of pixels and with each additional pooling tracking the original positions of the remaining pixels becomes more and more difficult for the model. This makes pooling the main reason for the much needed translation invariance in convolutional networks[14].

Architectures of simple convolutional networks involve a series of alternating convolutional and pooling layers, followed by fully connected layers. Their input is a flattened vector representing features detected in the previous layers. The fully connected layers work similarly to the traditional neural networks. Based on the combinations and the strengths of each detected pattern, they assign probabilities of the image belonging to each class through the softmax activation function. Softmax is an activation function suitable for the final layer of networks for multi-class classification. It is a function tailored to translate class logits into probability values with a total sum of 1[41].

The process of the image passing through each subsequent layer until the classification scores are assigned is called a forward pass. A model that has finished training and has been deployed for practical usage, only requires the forward pass to fulfill its purpose. During training however, each forward pass must be followed with a backward pass that implements the idea of backpropagation. Backpropagation's goal is to adjust each trained parameter (e.g. kernel weights, biases) to minimize the loss function. The parameters that are not trainable but are responsible for controlling the training process and the network structure are called hyperparameters. Hyperparameters influence the convergence speed and the final accuracy of the model. Learning rate is the most basic hyperparameter in neural networks and it is responsible for how quickly the model can reach a local minimum of the loss function. Lower values of this hyperparameter are preferable to prevent 'overshooting' past the optimal loss function value[14].

Convolutional neural networks are a more specific case of neural networks and this is why they require additional hyperparameters for controlling the process of adjusting the trainable parameters. Each convolutional layer consists of a given number of n x n sized

kernels that replace the traditional neuron weights and each value inside each kernel is a trainable parameter. The size of the kernels (also called filters) across the computer vision field is the most commonly set to 3 x 3 (7 x 7 is also sometimes used for the first convolutional layer). 3 x 3 kernels are the smallest kernels that are capable of retrieving patterns using the neighborhood information from input data. The approach of using the smallest possible kernel size is comparable to using fewest neurons possible in a normal neural network with multiple hidden layers. Theoretically, adding an additional hidden layer in a neural network is comparable to adding an exponentially increased number of neurons. This way, having a small number of neurons across multiple hidden layers is highly preferable to having one hidden layer with an excessive number of neurons. Each consequent layer is also capable of achieving a higher level of abstraction that normally would never be possible with only one hidden layer[16]. This pattern can also be observed in convolutional neural networks under the concept of receptive field.

The receptive field is the region of an input image that affects the output of a neuron. Its size determines whether a network focuses on patterns that are more granular or global, which should be adjusted considering the nature of the input data. The primary method of increasing the receptive field is adding extra convolutional layers. Considering a convolutional neural network with 3 convolutional layers, each using a 3 x 3 kernel, one neuron from the third layer will have a receptive field of 7, which means that the third layer neuron will be influenced by the data from the pixels inside a 7 x 7 pixel square. It might be important to highlight that the most influential pixels for any neuron belong to the kernel size from the first layer – if a 3 x 3 kernel is used in the first layer then those pixels will impact the neurons in the next layers the most, even if the overall receptive field is larger[23].

Increasing the number of layers is typically correlated with better model accuracy and longer training times[16]. Each subsequent layer can recognize higher level abstract patterns with exponentially more underlying patterns than in shallow models. However extensive tests show limited accuracy increases from adding layers past a certain point, while simultaneously drastically increasing training time. The most likely explanation can arise from intuitively understanding the feature maps of deeper layers of CNNs. The first convolutional layer extracts the low-level features of the image like shapes of the edges and curves. With each subsequent convolutional layer the complexity of each feature map increases, allowing it to

recognize complex shapes assembled from the individual patterns detected in the first layers. Past a certain point additional convolutional layers are no longer capable of abstracting from the previous layers which causes the feature maps to look nearly identical layer to layer. However, very deep convolutional networks become a necessity as the scale of the network increases. This scale can be represented using three dimensions[42]: depth, width and resolution. Image resolution used for training helps avoid losing the original information. If the model needs to capture very complex shapes, a very high resolution might be needed. The width stands for the number of filters (and therefore channels) used in each layer. Increasing the number of filters allows the model to capture more fine-grained patterns while also not increasing the training time drastically. This still does not increase the level of abstraction understood by the model nor the receptive field needed to perceive large objects in the image. For this purpose additional convolutional layers are added to the architecture, increasing its depth (as long as the vanishing/exploding gradient problem in very deep networks is accounted for).

In certain applications it makes sense to scale the model across one dimension – this was the standard method used when designing architectures like the deep ResNet, the wide WideResNet and the high-resolution Inception. A better approach might be to perform compound scaling by increasing each dimension uniformly[42]. This approach allows the convolutional networks to maximize the ratio between model complexity (number of parameters) and its accuracy.

$$\text{depth: } d = \alpha^{\phi}$$
$$\text{width: } w = \beta^{\phi}$$
$$\text{resolution: } \gamma^{\phi}$$
$$\text{s.t. } \alpha * \beta^2 * \gamma^2 \approx 2$$

The method requires performing a standard hyperparameter search in order to determine what is the most efficient way to increase each dimension with a very precise computational-cost limit controlled by the φ value. Each dimension is tied together with coefficients which allows to scale them in a balanced way (Figure 2.3) instead of just scaling the model in one particular dimension[42]. The most computationally expensive operation in convolutional networks is the convolution itself so increasing the width and resolution increases the computational cost quadratically, as opposed to the depth increasing it linearly.

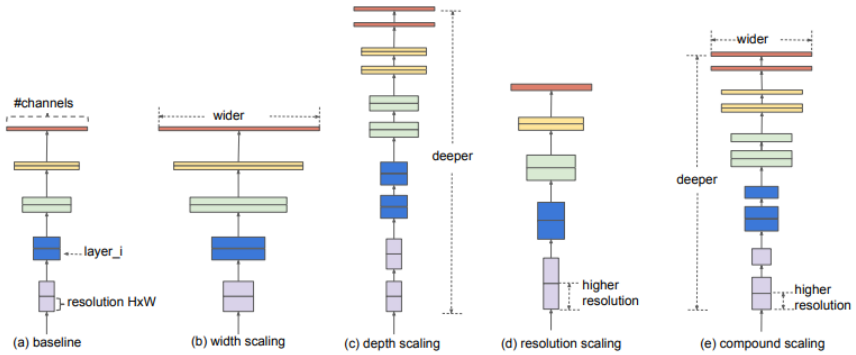

Figure 2.3. Approaches to scaling CNN architectures. Reprinted from [42].

2.3. 3D convolutional networks

In line with the goals of the thesis, two architectures have been chosen to be tested with the created dataset. The choice was dictated by the aim to capture the popular and established approaches to analyzing video data – 3D convolutional networks and Video Transformers. The main architectural approach that was omitted due to time constraints was the dual-stream network. To this category belong established models like:

- Two-stream CNN[33] – architecture combining a spatial 2D CNN RGB stream pretrained on ImageNet dataset, with a temporal 2D optical flow CNN stream as shown on Figure 2.4.

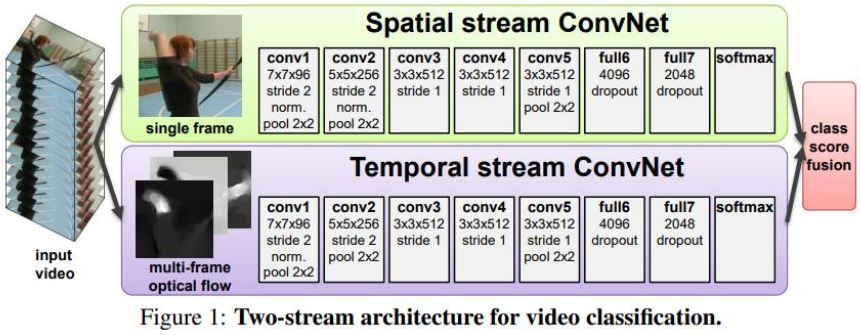

Figure 1: **Two-stream architecture for video classification.**

Figure 2.4. An example architecture of a dual-stream CNN. Reprinted from [33].

- I3D[6] – an architecture with two streams, created by taking two ImageNet-pretrained Inception-V1 2D CNNs and turning their 2D kernels into 3D kernels by repeating weights along the time axis (kernel inflation). One stream takes RGB as input, and the second stream works with optical flow.
- D3D[37] – network utilizing optical flow teacher stream to train student stream to work alone during inference using only RGB input. The teacher stream is only used during training to distill motion understanding from optical flow to the student stream.

Student loss is calculated by using standard classification loss and adding distillation loss that represents the difference between student and teacher classifications. This approach allows to significantly lower inference cost.

One of the main things that need to be accounted for is the used representation of video data. The naive approach of feeding RGB data into the network can potentially lead to satisfactory results but for achieving maximum accuracy other approaches need to be tested or combined together[6].

3D convolutional networks were popularized in 2015 by the paper 'Learning Spatiotemporal Features with 3D Convolutional Networks'. The paper proposed a simple and efficient approach for handling video data. Convolutional networks are known for being computationally cheap, and the usage of 3D convolutions allowed to train computer vision models without performing more manual methods of feature extraction from videos. Using 3D convolution kernels is a very simple and natural way to work with data with spatiotemporal characteristics[43]. 3D convolutions are significantly more computationally expensive than 2D feature extraction with temporal pooling but they capture deeper spatiotemporal patterns.

Before 3D CNNs and two-stream networks became dominant, 2D CNNs were tested on video data. A necessary addition was stacking multiple consecutive video frames in a multi-channel stack to be fed into 2D CNN as input. As shown on Figure 2.5, this allowed the network to work with a meaningful temporal perspective, albeit very limited to several subsequent frames, requiring noticeably longer training times and still achieving only minimal understanding of temporal relationships. In order for 2D CNNs to become effective, they need to be used along with mechanisms like optical flow, depth maps, LSTM or dual-stream architectures**Error! Reference source not found.**.

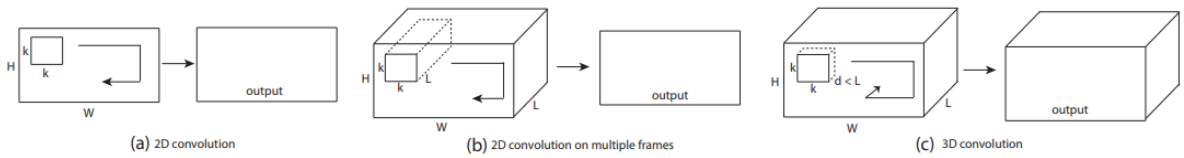

Figure 2.5. CNN's ability to preserve temporal relationships. Reprinted from [43].

As visible on Figure 2.5, the main difference between using 2D CNNs with frame-stacking and 3D convolutions is the type of layer output. While all data in modern vision networks is always processed as 2D stacks of channels, 3D convolutions allow to preserve temporal

structure of videos in the next layers, which gives them many more opportunities to extract more temporal context[43].

2.4. Video Transformers

Transformers are a family of architectures designed initially for NLP tasks that became popular in 2017[46]. They are based on mechanism called attention that serves as a compatibility function between an encoded input (query *Q*) with the related concepts or patterns (keys *K*) and finally with proposed outputs (values *V*). In NLP Transformers can be used to predict the next token that fits based on all the previous tokens. Due to the use of softmax the Transformer returns a list of probabilities of the most fitting answer (value *V*). In the Large Language Models the temperature parameter dictates how often less likely values will be returned – temperature of 0 will always return the most probable value.

A common attention function is dot product (visible here as matrix multiplication of QK^T) – it is a formula that is part of cosine similarity used to determine how similar two vectors are. The dot product is often scaled down (here using the key embedding size d_k of the current attention head) to prevent weights from exploding[46].

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) * V$$

Transformers are largely based on the Encoder architecture that is used to encode the input $(x_1, \dots, x_n)$ into an embedding vector $z = (z_1, \dots, z_n)$ with embedding size of n, using learnable weights.

Transformers were adapted to work with image in 2020. The paper 'An Image Is Worth 16X16 Words: Transformers For Image Recognition At Scale' proposed an approach to effectively encode image x by splitting it into 16 x 16 patches x_p (Figure 2.6). Normally to calculate self-attention for an image, each pixel would require separate attention calculation which would scale the cost quadratically. By combining pixels into patches much less computations are required. Preserving the spatial structure of an image is possible thanks to positional encoding achieved by fully learnable linear projection of the sequence of patches – one common set of weights is used to encode every patch. As the result of linear projection, the image is represented as a 1D vector of 1D projected patch vectors, each embedded with embedding size of *n*. On top of that, to that embedding vector a class token (called CLS) is appended. The CLS token is randomized during input and is used by the Transformer to store

general information and representation of the whole image. The CLS token is modified between each layer of multiheaded attention and is used as the query *Q*, representative of the whole image[12][28].

Embedding size is used in Transformers to represent the input with complexity depending on *n*. Increasing embedding size allows to increase the amount of information that the Transformer can store during analysis of the input. All concepts learned by the Transformer are stored as embedding vectors that are grouped in the *n*-dimensional space and their Euclidean distance informs how similar each concept is to another[12].

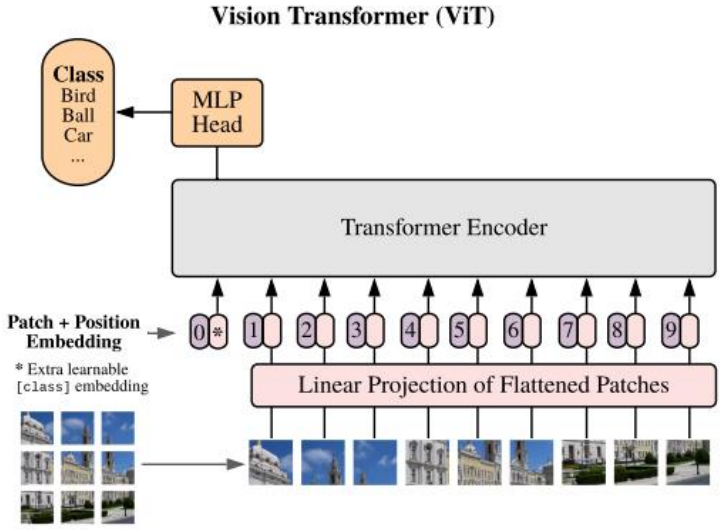

Figure 2.6. The architecture of ViT. Reprinted from [12].

Multi-head attention works by splitting the embedding vector *z* into *h* attention heads (e.g. a 512-dimensional embedding vector can be split into 8 attention heads as 64-dimensional vectors). Each attention head can be perceived as a much more advanced and versatile convolution kernel, looking for a specific set of features using separate learnable weights to store information. The result of attention for each head is then concatenated into an embedding vector of the original embedding size *n* that represents the combined extracted features from all attention heads[46].

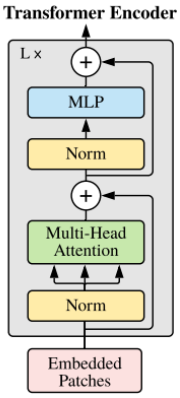

Figure 2.7. Encoder of the ViT. Reprinted from [12].

In Vision Transformers, attention can be visualized by taking the output value *V* (the assigned class label) and by seeing how attention scales the original query Q (the image). This is visible on Figure 2.8, where for each class label it is possible to visualize which parts of the input were used to make the decision. Considering that attention scales each dimension of the embedding vector depending on its importance to each key *K*, in order to preserve the original positional encoding residual connections (denoted as + on Figure 2.7) are used to combine the original embedding vector with the scaled embedding vector[46].

Figure 2.8. Visualization of attention in ViT. Reprinted from [12].

The reason why the Transformer became the most popular modern architecture is in large part due to its immense parallelization potential. Compared to recurrent networks that are dependent on the previous steps, Transformers can process an entire sequence at the same time. Classification tasks that are focused around the Encoder can be performed as a single, large-scale matrix multiplication that returns the attention result and therefore, the classification label[46].

Timesformer[3] is an adaptation of the ViT extended to work with video. The approach is largely the same and the sequence of frames is represented as a 2D tensor that is a concatenated sequence of frames embedded analogically to ViT. Spatiotemporal relationships are preserved through adding the spatiotemporal positional embedding $e^{pos}_{(p,t)}$ to each embedded patch *z*, where *p* denotes patch index and *t* depicts number of the frame.

$$z^{(0)}_{(P,t)} = Ex_{(p,t)} + e^{pos}_{(p,t)}$$

For layer *l* and attention head *a* the current query can be encoded by performing normalization *LN* on the embedding vector of patch *(p,t)* returned by the previous layer *l-1* and multiplying it by a set of weights W_Q unique to the current layer and attention head. Calculating key *k* and values *v* for each patch *(p,t)* in each layer's attention head *(l,a)* is done analogically[3].

$$q^{(l,a)}_{(p,t)} = W^{(l,a)}_{Q} LN\left(z^{(l-1)}_{(p,t)}\right)$$

$$k^{(l,a)}_{(p,t)} = W^{(l,a)}_{K} LN\left(z^{(l-1)}_{(p,t)}\right)$$

$$v^{(l,a)}_{(p,t)} = W^{(l,a)}_{V} LN\left(z^{(l-1)}_{(p,t)}\right)$$

There are multiple possible ways to compute spatiotemporal attention in Video Transformers. The best performing method in the Timesformer paper is Divided Space-Time Attention that works by layering two attention steps. The first step calculates temporal attention between all patches with the same position *p* but different frame index *t*. Using residual connection, space attention is performed next between all patches with the same frame index *t* but different positions *p*. Then another residual connection is used to combine the initial positional encoding with both attention types. Then a Multi-Layer-Perceptron *MLP* sublayer is used to modulate the embedding vector further and to apply non-linearity to it. The process is illustrated on Figure 2.9[3].

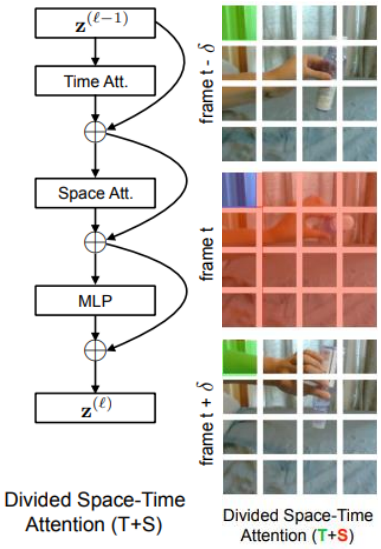

Figure 2.9. Divided Space-Time Attention in the Timesformer. Reprinted from [3].

Transformers belong to the largest used deep learning architectures. While they train quickly despite their size, they still require large volumes of data. Transformers are capable of internalizing an immense number of patterns but they are very prone to overfitting smaller datasets. State-of-the-art accuracy can be achieved on most datasets provided that transfer learning from biggest datasets is utilized[1].

3. Generating and testing synthetic data

3.1. Synthetic video clip generator

In the thesis the approach of synthetic data generation was tested and as a result, a Unity application for generating dynamic camera shots was created (Figure 3.1). It is fully functional and possesses mechanisms for preventing incorrect video generation while also featuring additional tools for making dataset creation smoother.

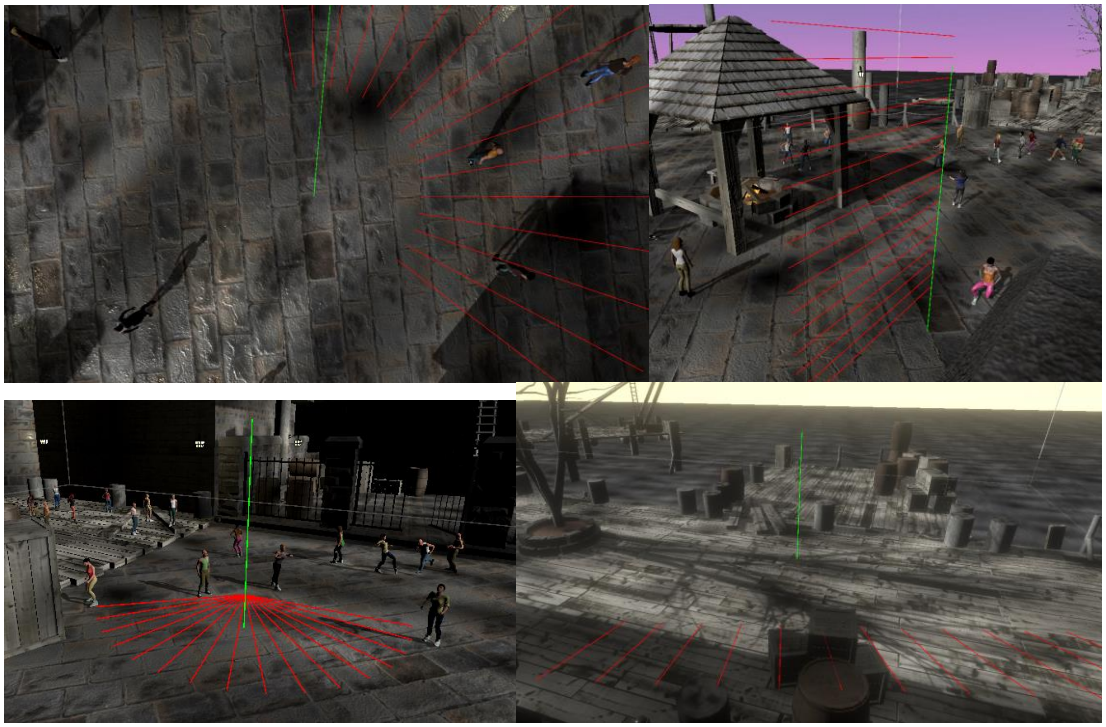

Figure 3.1. Visualization of different shot types on map Port. a) arc, b) vertical dolly, c) pan, d) sideways dolly. The green, vertical line represents the recorded interest point. The red line represents every camera position and angle during the shot.

Currently 9 types of camera movements (in practice 17 when including both possible directions of a movement) can be generated using the video generator. The shot types were designed to include most of those used in cinema[2][38]:

- Static – the camera does not move.
- Arc – the camera orbits around the recorded point.
- Pan – a panoramic shot made by rotating the camera from a single position.
- Roll – a chaotic rotation along the long axis of the camera – the camera does not move.
- Lateral dolly (called sideways in the tool) – the camera is moved steadily left or right.
- Vertical – the camera is moved steadily up or down, not changing the shot angle.

- Zoom – the camera does not move but the FOV (Field of View) change makes the scene appear to move closer or farther from the camera.
- Push – the camera moves steadily towards or away from the recorded object.
- Dolly zoom – this shot combines both zoom and push types, causing the center of the recorded frame to move dramatically away or towards the audience.

The tool is integrated with 8 environments that were free assets on the Unity's Asset Store at the time of working on the project. Each of the maps (Figure 3.2) has been manually adjusted to work with Unity's *NavMesh* system[44] for pedestrian pathfinding.

In order to provide the maps with more variety and moving elements, a pedestrian pathfinding system was implemented using Unity's AI package. The system requires adding a *NavMesh* Surface component to a top-level *GameObject* and restricting the walkable area by applying the *NavMesh* Modifier component to each object or surface where pedestrians should not walk.

Figure 3.2. Example environments in the generator. a) Space Station, b) Dungeon, c) Forest, d) Mars. Varied lighting conditions can be observed.

The logic of pedestrian spawning and pathfinding is performed inside the *PedestrianSpawner.cs* script. To ensure the variety of pedestrian appearances the free UMA asset (Figure 3.3) was used that works as a versatile, procedural-based system for defining

possible variations of character models and their wardrobe combinations. While the system allows for real-time character generation, the models have been pre-generated in the number of 100, which significantly speeds-up clip generation. In future version of the generator the number might be decreased as the variety is not worth the required disk space of 0.5 GB. The real-time generation is available as well, as the UMA API was integrated into the *PedestrianSpawner.cs* allowing to optionally enable the Dynamic Character Creation. The experience with UMA during the project was generally unpleasant and due to a significant number of bugs on newer versions of Unity the asset is not recommended to be used in similar projects. One of the glaring issues is the requirement to reload UMA after every project modification.

Figure 3.3. UMA demo scene. The asset is also available to use through API.

Pedestrians can be spawned in two versions – idle or running. Idle pedestrians are randomly assigned with one of 3 animation groups after spawning – sitting (currently not used), standing or the *Uncombinable* group containing animations that are performed indefinitely as they would not work well when transitioning to another animation. The standing animation group consists of animations that are chosen at random with randomly multiplied speed every time the animation cycle ends. The animations were all taken from *Adobe*'s *Mixamo* project or from Unity asset packages and they are all currently free to use. The generator currently uses 57 standard idle animations (e.g. hand gestures or jumping), 14 *Uncombinable* idle animations (e.g. bow shooting or fist fighting) and 9 running animations (e.g. sneaking, walking or running).

Pedestrian is generated using *PedestrianSpawner.GetValidRandomPoint()* that finds a random spawn point on the walkable parts of the map, with a minimum specified distance away from all the other currently spawned pedestrians. The candidate spawn points are validated using the *NavMesh.SamplePosition()* function that returns the closest valid point on the *NavMesh* to the proposed spawn position. The same point sampling method is used for randomizing running pedestrians' destinations using *NavMeshAgent.SetDestination()*.

Varied lighting conditions can optionally be augmented using built-in post-processing methods like bloom, vignette, depth of field focus, chromatic aberration, saturation and contrast with parameters randomized in every shot. The lighting is also modified in most maps by randomizing the skybox from the pool of 8.

For purposes of camera movement classification, the most important kind of variety in this generator comes from the parameterization of the camera shots. Due to the major differences between each map (especially related to their size) each map stores a separate parameter set inside its own copy of the *EnvironmentSettings* script. The shots are randomized using parameters like:

- Distance from the interest point (randomly chosen, recorded point of the map) – needed to spawn the camera for all shots.
- Travel distance – required for sideways and vertical shots.
- Camera height – relevant to pan and vertical shots.
- FOV (Field of View) – modifies all shots except zoom, where it is overwritten by more specialized parameters.
- Rotation angle – describes the range of arc, pan and roll movements.

Creating a complete dataset from the offered features is relatively time consuming because of the need to manually adjust each shot type parameter for every map separately. However, after the parameters have been determined the generator is capable of generating most shot types on all maps in a single long session. This can be done by choosing every environment inside the *GlobalVars* script and every shot type in the *SettingsManager* script – both of the scripts can be found in the *GlobalSettings* object in the scene's *GameObject* hierarchy (Figure 3.4). Currently push and sideways shots should be done separately from the others as they are unstable and can cause memory leaks due to incorrectly deleted *GameObjects* during shot validation procedures.

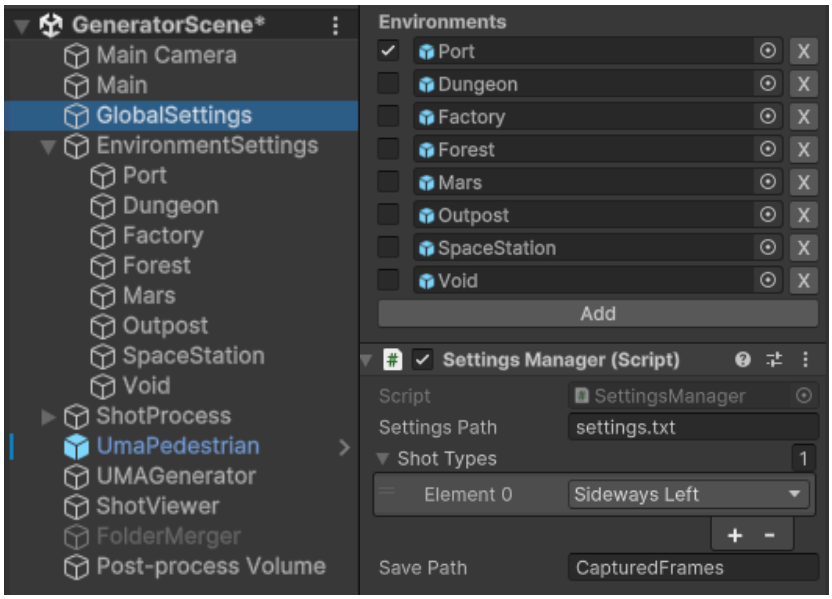

Figure 3.4. Choosing environments and shot types for a recording session.

All settings and generated information are stored using JSON files. All major procedures are logically separated by cycles of saving and loading JSON information, which allows to deterministically reproduce most of the scene (with notable exceptions of: post-processing, pedestrian behavior and skybox choice as seen on Figure 3.5). While the current version requires using the Unity Editor to access, the design choices allow to create with little effort a future final build version working purely with JSON configuration files. Current version of the generator stores JSON information for:

- General settings – including the save path, time scale, video length and resolution.
- Shot plans – a self-explanatory example on Figure 3.5.
- Each frame – allows to reproduce camera position and behavior for future research.


```
{
  "arcDegrees": 132,
  "tiltAngle": 5.0,
  "id": 0,
  "environmentType": 0,
  "fps": 8,
  "duration": 2.0,
  "FOV": 42,
  "shotType": 2,
  "observedPoint": {
    "x": -25.0464458,
    "y": 11.7986841,
    "z": 74.51032
  },
  "cameraLocation": {
    "x": -25.2931442,
    "y": 19.0289936,
    "z": 67.50085
  }
```


Figure 3.5. All information needed to reproduce an arc shot.

The general process of performing a shot is illustrated on Figure 3.6. The most notable part of the system is the *VisibilityTester.cs* script. If a shot is deemed invalid then after finishing processing every shot plan, replacement shots will be generated to match the specified number of clips. The script itself is a combination of multiple techniques for predicting the outcome of the recording and controlling quality in real time. It involves methods like:

- Spawning a ghost camera object before recording commences – a physical object is needed and is moved in the scene to check if camera will collide with the surroundings. The collider box's size can be modified by the user or even set to none.
- Checking for pedestrian occlusion (optional) – since the pedestrians are a valuable source of randomness a shot not including any pedestrians could be viewed as invalid. To check for pedestrian visibility only raycasting is currently performed but frustum culling AABB check should also be added to verify if pedestrians are actually in the frame.

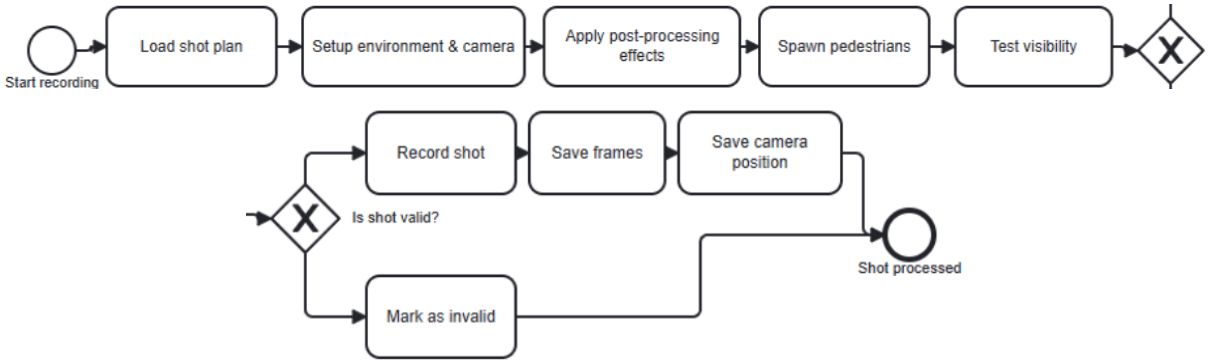

Figure 3.6. The process of recording a single shot visualized using BPMN notation.

Procedural generation in complex scenes is extremely difficult to fully test. Since the used maps were provided by external sources, they are not fully compatible with the considered task. This inevitably leads to the need for manual review of each generated clip. The proposed solution (Figure 3.7) works by showing the user 4 representative frames for each clip and allows to remove any of them renumbering the resulting directories. The current weaknesses of this approach include lack of solution for reviewing class label accuracy – 4 frames per clip are not enough to determine if the label corresponds to human intuition. An actual video browser functionality might be needed for this. There is also no automated mechanism for replacing the faulty clips with newly generated ones.

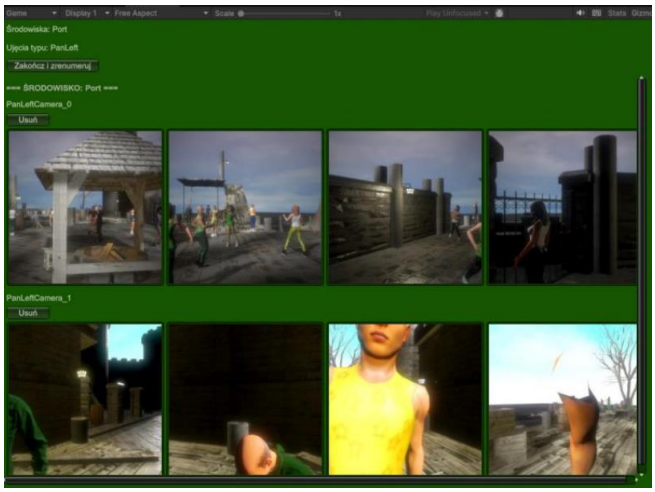

Figure 3.7. Post-recording clip overview.

Computational efficiency was also considered during the system design. Batch recording (multiple cameras recording at the same time) has been tried but proven to be ineffective. It is because pixel processing during output saving is responsible for most of the compute time – this can be confirmed when increasing the resolution of the clips being saved and it is not related to the speed of the disk storage. The fact that the rendering and scene calculations are less expensive than retrieving and saving frame pixels as a JPG 95% texture, warrants further investigation into the efficiency of the current approach to retrieving and saving pixels.

A known problem with the generator is the possibility of memory leak whenever shot is cancelled after predicting camera collision with the terrain. This happens often on some of the maps (Dungeon, Factory, Space Station) while generating push and sideways shots. The exact cause is not established but might be due to *GameObjects* not being destroyed properly after detecting camera collision with terrain. For this reason, the best way to generate these shot types is using smaller batches.

3.2. Synthetic dynamic camera shot dataset

The model of reality inside the generator's logic can be observed and evaluated using the prepared dynamic camera shot dataset prototype. The presented dataset is not a final proposition due to the observed limitations of the current version of the clip generator:

- Clip validation methods need to be improved, especially related to pedestrian visibility in a frame.
- Life quality features should be added to make the unavoidable manual review process quicker.

- Two additional shot types need to be added – vertical pan and irregular handheld.
- Camera position-choosing logic must be improved – e.g. preventing arc shots from being recorded directly above the recorded point – such shot is indistinguishable from a roll shot.

In order to ensure that the dataset is fully balanced regardless of the scene environments chosen for training, each environment was used to generate the exact same number of 300 clips per shot class. The number intuitively in relation to each map's size might be the highest possible when optimal variety between frames is desired. However, increasing the number of generated clips per class would not cause models to overfit easily thanks to the used techniques of procedural generation. Despite that, due to the considered problem's simplicity (understanding a motion pattern was observed to be relatively easy for various models), it might be a better idea to not increase the number significantly. The exact specifications of the dataset are visible in Table 3.1 and the minimum guaranteed clip validity of an environment, measured by the test validation accuracy on a given unseen environment, is shown in Table 3.2.

Prototype of the dynamic camera shot dataset - parameters							
Total number of clips	Number of classes	Number of clips per class	Number of environments	Frames per second	Clip duration	Disk space taken	Extra data modalities
36,000	15	300	8	8	2 s	4.89 GB	Camera position of each frame

Table 3.1. Parameters of the dynamic shot dataset.

The process of refining the dataset parameters involved multiple steps of training various architectures (specifically C3D and Timesformer) using a given iteration of the dataset. The dataset cannot be seen as complete yet because it has not been fully reviewed manually – it should be the final step of dataset refinement that is currently pointless and inefficient due to the shortcomings of the generator.

Without a detailed manual overview of the dataset, it can be estimated that for maps: Port, Outpost, Factory, Dungeon and Forest the percentage of valid clips is a figure of at least 87%. This can be observed both from the accuracy while training smaller models that are

unable to overfit the data, as well as during generalization tests for unseen environments. This generalization test method can be seen as a variation of the traditional cross-validation method well-established in the ML history. Testing the model on a new map allows to check for overfitting and to automatically control the current dataset iteration's quality.

Another objective during the dataset creation was achieving adequate model accuracy with minimal required disk space and maximal processing speed to accelerate the prototyping phase. During quick tests on a slimmed-down C3D model it was observed that changing the clip length from 72 frames to 16 did not affect the test accuracy of the models despite reducing the training time by a factor of 4. 16 frames is also a common value for input clip length in HAR tasks. Resolution was also reduced from 256 x 256 to 128 x 128 through this process. 64 x 64 was also tested but affected the model accuracy significantly. In the final version of the dataset a resolution of 256 x 256 will be considered as it seems detailed enough to the human eye. For comparison, see Figure 3.8 for an example of a 128 x 128 resolution clip. Despite anti-aliasing being used, the rough edges are very visible and make it difficult to tell the exact shape of the pedestrian models.

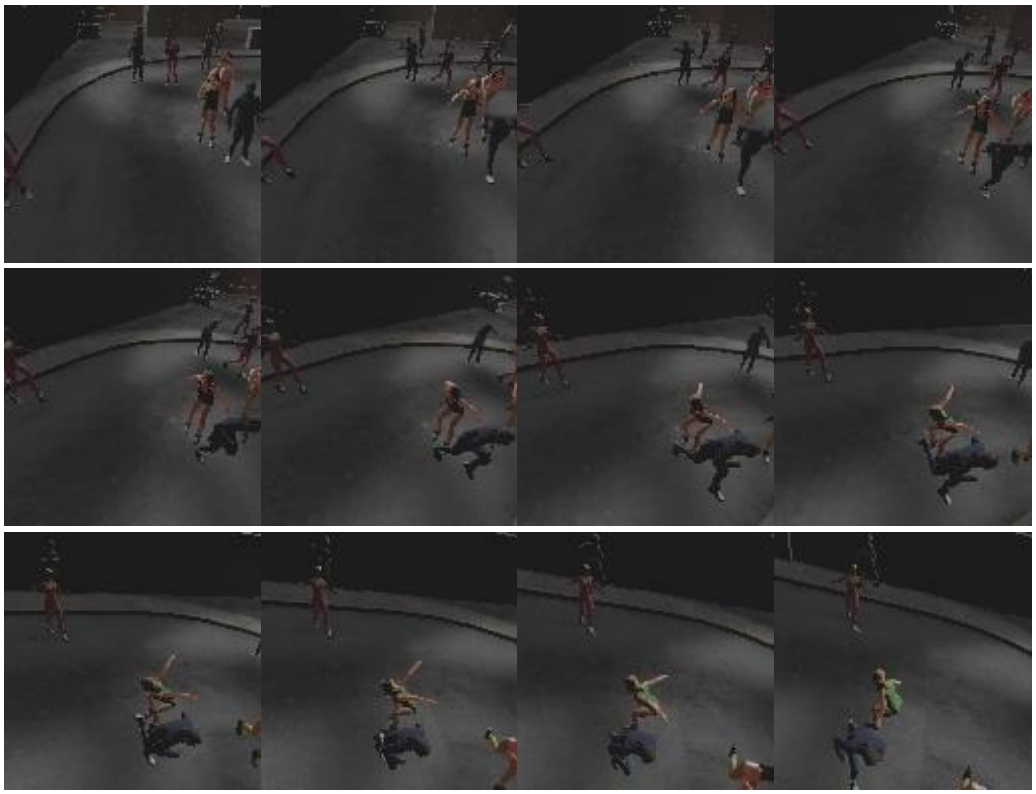

Figure 3.8. An example of a generated counter-clockwise arc (ArcRight) clip on the Factory map.

The current parameters make the dataset easy to use for educational purposes, as it can be quickly downloaded to train small and efficient models. The taken disk space could still be reduced by a factor of 2 by storing the clips as encoded video files instead of individual frames. The complete dataset might be released in 2 or 3 variations depending on the intended application. Videos with increased resolution and length might prove to be useful for training deeper architectures and they would fare better in tasks like flow and depth estimation or HAR. Additional data modalities like pedestrian information are also intended in the future releases.

The minimum guaranteed clip validity							
Environment	Port	Mars	Factory	Forest	Outpost	Dungeon	Averaged
Guaranteed clip validity	92%	93%	88%	81%	90%	78%	87%

Table 3.2. Estimation of the clip validity of the selected maps. The metric used was test validation accuracy.

3.3. Tests with the use of synthetic data

The Timesformer architecture[48] was tried on the dataset to assess the potential required number of clips to train a large-scale architecture. Despite using very small values of hyperparameters of: embedding size = 256, number of attention heads = 8 and depth = 4, the 7 million parameter model overfit the current dataset very quickly with 98% training validation accuracy, but only 80% testing validation accuracy. Although more clips can be added to prevent overfitting, a transfer learning approach with publicly available Kinetics-400 pre-trained models is more recommended. An approach with the I3D dual-stream architecture should also be tried due to its high results in similar studies.

Focusing on immobile background elements seems to be a valid method for approaching classification of dynamic shots and it probably is the method used intuitively by humans. To maximize robustness and the generalization ability of models in this application, deeper focus must be put on making backgrounds more varied. To test the importance of backgrounds in the process of understanding an additional environment called Void was created. It features a blank space and the same skybox pool as other environments. While the model was able to achieve high training validation accuracy during training on that environment, the knowledge generalized very poorly to the Mars environment, achieving only 50% test validation accuracy.

During creating the dataset one particular thing was observed. For clean distinguishing between class types, the camera needs to have enough maneuver space. Environments with small corridors are a poor source of dynamic camera shots because they do not allow for full scope of motion. Incomplete (e.g. due to lack of space) shots blend together class-wise and their classification might even become impossible, thus bottlenecking the upper bound of obtainable accuracy. This limitation can be overcome by scaling the size of maps with narrow areas. It must be noted that changing map size requires adjusting lighting intensity and rebaking the *NavMesh* for pathfinding purposes. Scaling the map is a simple and effective way to increase clip variety as long as the map is detailed enough. Unfortunately, this method is not the intended way to use *NavMesh* in Unity and scaling the map in any way causes pathfinding to encounter issues.

Results achieved using the C3D 3D CNN architecture showed that the expensive optical flow calculations are not needed for achieving acceptable results. While the tests utilizing the optical flow were not performed due to the early stage of the dataset development, it is possible that they could achieve better results.

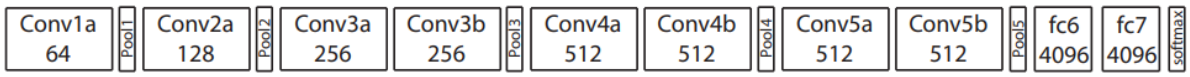

Figure 3.9. Diagram of the C3D architecture that performed the best on the prototype dataset. Reprinted from [43].

Two version of C3D were tried – one version being the original one used in the C3D paper, and the other being a trimmed down version without the Conv5a and Conv5b layers. This version allowed the models to train twice as fast with only 1 percentage point test validation accuracy loss.

The map chosen intuitively as the map with the highest quality of clips was Mars and it was used as the test validation data subset, with maps: Port, Outpost, Forest, Dungeon and Factory used for training. With this approach and the Adam optimizer with learning rate 1e-4, the full C3D model achieved 93% test validation accuracy after 20 epochs, and the reduced version achieved 92% test validation accuracy on Mars after 15 epochs. The values can be improved by refining the data generator. The achieved results suggest that the generated synthetic patterns are not challenging for modern architectures.

The adopted class definitions and division warrants further deliberation. Rare classes like dolly zoom might not be worth keeping in the model, especially considering the similarity to push and zoom shots. Even the distinction between left-hand and right-hand movements might not be needed at all (unless it would be used to show possible bias of movie directors to one-side camera movements). Another particular case of conflated classes is arc shots with camera spawned directly above an interest point – this causes the arc shot to become identical with roll shot. This can be revised by revisiting the rules for choosing camera spawn points – currently camera can spawn anywhere in given radius as long as the spawn point is above ground level (detected using raycasting). The surface of valid spawn points forms a half-sphere (assuming the ground is even). Spawn points could be restricted for specific shots – e.g. arc shots should not be filmed from directly above. Another case where accuracy does not tell the full story is the distinction between zoom and push shots. Depending on camera spawn point and the movement scale, both classes could be impossible to tell apart.

As can be observed on Figure 3.10, the created generator and dataset prototype struggle with defining distinctions between classes like:

- Arc and roll – this comes from the flawed camera spawn point generation, where the camera might spawn directly above the observed point and will not be able to perform an orbiting movement. The camera cannot orbit around the observed point from that spawn position, which will effectively turn the orbiting arc movement into immobile roll movement. Most of the confusions come from this shortcoming of the current camera random spawn system.
- Zoom and push – while the two motions are very distinct, they might blend together whenever the background information is lacking. The class definitions

used in the generator are correct and it might not be possible to automatically rectify the ambiguity between edge cases in class instances.

- Sideways and pan – unexpected but the issue is caused by the used parameters during generation of these shots. Sideway shots need longer travel path and pan shots need to perform larger rotation in order for them to be clear and recognizable.

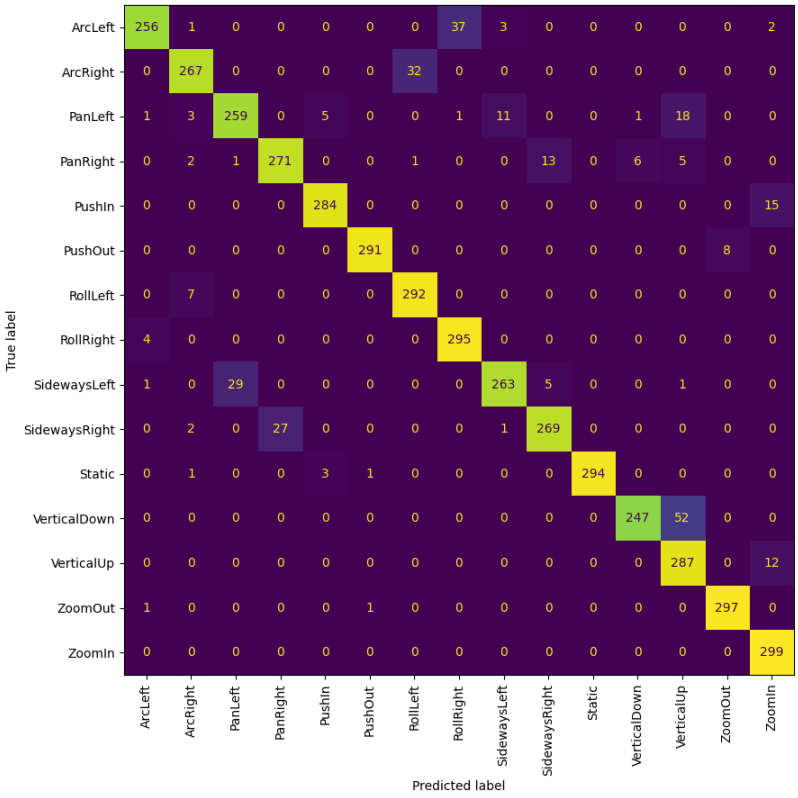

Figure 3.10. Confusion matrix of the full-sized C3D model tested on the Mars map.

The class-labeling dilemma leads to a currently discussed problem[26] with labels in datasets and their relationship with ground truth. Ground truth is hard to define between it being an amalgamation of objectively true labels and opinions of domain experts. A better way to look at the definition is to view ground truth as the labels we desire for the model to assign to each class instance. This causes issues whenever experts disagree on individual labels. With this in mind, ground truth could be defined as a label that at least a single expert would agree with. While the misclassifications between arc and roll are preventable and possible to be removed in future versions of the dataset, the zoom and push distinction (with possible dolly zoom as well) might be impossible to fully overcome as they are very similar in practice.

The lack of real-life-data testing makes it difficult to evaluate the dataset prototype with regards to class-modeling accuracy. It is possible that camera movements in the dataset are exaggerated compared to actual cinematic shots and too easily discernible. If real-life video is used to test the dataset quality, it will become possible to adjust the camera movement strength to more realistic levels. In case that the current shots are too clear-cut, camera movement strength will need to be reduced and the resulting drastic increase in number of ambiguous shots will have to be addressed manually.

The used method for detecting if camera spawn point is above the ground level (raycasting downwards) has an unexpected weakness when used on the map Space Station. The map is very complex and does not come with precisely mapped object colliders which can lead to situations where arc shots are filmed upside down causing them to become mirrored, changing its class. The map layout also makes it difficult to obtain good quality footage. For this reason the environment's clips were not included in the tests.

Due to existence of large-scale datasets like Kinetics-400 that can be used to drastically reduce the required volume of target-domain data via transfer learning, newer datasets tend to be smaller, more specialized and higher quality. While some noise is unavoidable and even positive, problems like class noise and data bias can impact model robustness regardless of how advanced deep learning architectures are becoming. For this reason dataset refinement is showing itself to be one of the most noticeable trends in computer vision. One technique worth noting is data sampling as a tool for reducing impact of ambiguous labels. It works by assigning weights *W* to the training samples in order to prioritize cleaner data in the learning process, which prevents the model from internalizing undesirable patterns[47].

Negative test cases (*the Ugly*)	
245	Cloud of visible particles (*e.g.* pollen, small leaves) in the air are obscuring the whole scene
504	Highly transparent object encompassing a second opaque object that gets distorted due to the other object's shape
695	Scene contains a large concave mirror that shows an clean upside-down copy of parts of the scenery
719	Observer is placed between two parallel mirrors facing each other so that "infinite" number of reflections occur
790	Left and right image are the same while showing a diverse scene
916	One camera lens contains dust/dried mud that creates a partially defocused area in the image
921	Lens is broken cleanly leaving a visible crack in the image's center
933	Images contain rolling shutter artifacts
955	Images contain considerable chromatic aberration and many visible edges
983	Images have considerable amounts of vignetting and scene contains many objects close to the observer
1094	One of the two sensors is somewhat out of focus
1105	Inter-lens reflections create visible copy of objects in the image
1162	Image before rectification originates from considerably rectangular pixels (instead of square, near to e.g. 2:1 ratio)
1166	Images contain strong static image noise for well-lit scenes
1261	One camera delivers negative image (or color channels swapped)

Table 3.3. Examples of data noise sources in autonomous driving. Reprinted from [49].

Noisy data can be beneficial in certain applications. E.g. in autonomous driving very unfair or illogical situations can arise depending on the weather, light conditions and surroundings. Despite the impossibility of linking such input data to the desirable behavior or label, the illogical and noisy data can shape the model in positive ways by instructing them how to act in extreme conditions. As shown in Table 3.3 autonomous driving models need to be able to handle all sorts of anomalies in order to ensure their safety[49]. Comparably, data generated by the generator created for this thesis can be difficult to interpret depending on where camera is placed relative to the scene. The Space Station map has multiple floors and many protruding elements that can obstruct camera vision. The important thing about handling this sort of scenarios is to manually review them and either remove them or change their label.

Conclusions

Real-life data for camera movement analysis is a scarce resource. The existing datasets, as described in the chapter 1, suffer from the overrepresentation of static shots or limited public availability. As demonstrated by the existing research[24][31] of dynamic camera shots, it is possible to train adequate DL models using only clips from real-life cinematic movies. However, this approach suffers from:

- Significant resources needed for data collection and labeling.
- Limited real-life instances of rare shot types like dolly zoom.
- Inability to publish the created real-life datasets due to copyright.

All of the mentioned shortcomings of real-life datasets can be remedied by the use of synthetic data, which is a data augmentation method recently growing in popularity[8]. As long as enough data variety is provided through well-designed procedural generation, a high amount of generalizable video data can be obtained through this method. Due to multi-modal datasets steadily gaining importance[1], synthetic generators might gain popularity themselves as the cheapest source of multi-modal data. This avenue was not pursued in the thesis but the implemented generator could be further developed with that in mind.

The most similar, existing solution for synthetic data generation is the driving simulator CARLA[13]. It is an open-source tool that provides a Python API that can be used to freely modify: the current map, scene and their objects. It implements mechanisms (e.g. weather conditions, pedestrians) that are a source of variety that can greatly improve the generalization ability of the generated data. While modifying this tool requires less resources than creating a new one, it is still limited to only urban environments. This could significantly affect the generalization ability of the generated dataset, which is a strong argument for implementing a custom solution.

Camera movement classification calls for methods capable of understanding temporal relationships between frames of a clip. This makes the problem similar to the Human Action Recognition (HAR) field. Among methods commonly used for those tasks are: 3D convolutions, dual-stream architectures and Video Transformers. Due to time constraints the dual-stream approach was not tested.

A well-established[22] method of representing motion in a video is optical flow. While including it into the workflow can increase accuracy of the trained models (especially with dual-stream architectures[6]), it is computationally expensive and is not necessary for training good quality models based on 3D convolutions or Video Transformers.

A synthetic video generator in Unity was designed and implemented for this thesis. The tool is fully functional, allowing to procedurally generate 9 types of camera shots (static, arc, pan, roll, zoom, dolly zoom, push, lateral, vertical) in 8 environments (Dungeon, Factory, Forest, Mars, Outpost, Port, Space Station, Void). To increase the quality of the synthetic data, methods were implemented for:

- Manual clip browsing and deletion.
- Real-time clip validity monitoring (e.g. detecting camera collision or counting visible pedestrians).
- Pedestrian appearance and movement variety.

While working on the generator, defining camera movement classes was found to be problematic. Future versions of the tool require adding vertical pan and handheld camera movement types. The generator also requires an improvement to camera spawn point randomization in order to prevent arc shots from becoming roll shots. Besides that, certain parameter values cause classes to blend together, which can happen for sets of classes like:

- Zoom, dolly zoom and push – class distinction depends on FOV (Field of View) ranges and the length of camera's trajectory.
- Pan and lateral – if the movement range is restricted they become very difficult to distinguish.

Experiments with the dataset parameters showed that the synthetic clips can be effectively used to train compact, accurate models with as little as 16 frames per clip and a resolution of 128 x 128. This allows the dataset to be used for educational purposes, with the current version taking only 5 GB disk space. With the current limitations of the generator, creating bigger datasets will not help improve accuracy of the trained models, as most of the class confusions are caused by insufficient quality of the generated data.

To ensure that the models trained on the synthetic data generalize well to unseen environments, an adaptation of cross-validation was designed. It was done by designating one environment (e.g. Mars) entirely as a test validation data sample. The best achieved model

accuracy (93%) was obtained by training a fully sized C3D[43] 3D CNN architecture on maps: Port, Dungeon, Forest, Outpost, Factory and testing the trained model on the Mars environment. This approach can also be used to automatize controlling the generated data quality, without requiring manual review. By averaging the results from this cross-validation method across all environments in the synthetic clip generator, a global estimate of 87% can be made for validity of the entire generated dataset.

References

[1] AlShami A. K., Rabinowitz R., Lam K., Shleibik Y., Mersha M., Boult T., & Kalita J. (2025). SMART-vision: survey of modern action recognition techniques in vision. *Multimedia Tools and Applications*, 84, pp. 32705–32776, (2025). https://doi.org/10.1007/s11042-024-20484-5

[2] Arijon D. (1976). Grammar of the film language. SilmanJames Press.

[3] Bertasius G., Wang H., & Torresani L. (2021). Is Space-Time Attention All You Need for Video Understanding? In: *International Conference on Machine Learning (ICML 2021).* https://doi.org/10.48550/arXiv.2102.05095

[4] Cabon Y., Murray N., & Humenberger M. (2020). Virtual KITTI 2. https://doi.org/10.48550/arXiv.2001.10773

[5] CARLA Simulator. Documentation. https://carla.readthedocs.io/en/latest/core_map/#environment-objects [1.09.2025]

[6] Carreira J., & Zisserman A. (2017). Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017).* https://doi.org/10.48550/arXiv.1705.07750

[7] Cauli N., & Recupero D. R. (2022). Survey on Videos Data Augmentation for Deep Learning Models. *Future Internet*, *14*(3), 93. https://doi.org/10.3390/fi14030093

[8] Cauli N., & Recupero D. R. (2024). Synthetic Data Augmentation for Video Action Classification Using Unity. In: *IEEE Access*, 12, pp. 156172-156183. https://doi.org/10.1109/ACCESS.2024.3485199

[9] Chen S., Wang X., Sun Y., & Yang K. (2025). STAN: Spatio-Temporal Analysis Network for efficient video action recognition. *Expert Systems with Applications*, 268, 126255. https://doi.org/10.1016/j.eswa.2024.126255

[10] Cubuk E. D., Zoph B., Mane D., Vasudevan V., & Le Q. V. (2019). AutoAugment: Learning Augmentation Strategies from Data. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019)*, pp. 113-123. https://doi.org/10.48550/arXiv.1805.09501

[11] Damen D., Doughty H., Farinella G. M., & Furnari A. (2021). Rescaling Egocentric Vision: Collection Pipeline and Challenges for EPIC-KITCHENS-100. *International*

Journal of Computer Vision, 130(9), pp. 1-23. https://doi.org/10.1007/s11263-021-01531-2

[12] Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., & Houlsby N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: *International Conference on Learning Representations (ICLR 2021)*. https://doi.org/10.48550/arXiv.2010.11929

[13] Dosovitskiy, A., Ros, G., Codevilla, F., López, A.M., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. In: *Conference on Robot Learning*. https://doi.org/10.48550/arXiv.1711.03938

[14] Goodfellow I., Bengio Y., & Courville A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org [1.09.2025]

[15] Hu J., You Z., Gu J., Zhu K., Xue T., & Dong C. (2025). Revisiting the Generalization Problem of Low-level Vision Models Through the Lens of Image Deraining. https://doi.org/10.48550/arXiv.2502.12600

[16] Huang G., Sun Y., Liu Z., Sedra D., & Weinberger K.Q. (2016). Deep Networks with Stochastic Depth. In: Leibe B., Matas J., Sebe N., Welling M. (eds) *Computer Vision – ECCV 2016*. *Lecture Notes in Computer Science()*, 9908. Springer. https://doi.org/10.1007/978-3-319-46493-0_39

[17] Kaplan M. A. (2014). Meta-Cinemetrics: Integrally-Informed Moving Image Metrics for Minimizing Risk, Maximizing Creativity and Profit, and Optimizing Viewer Experience. https://doi.org/10.13140/RG.2.1.2625.9922

[18] Kay W., Carreira J., Simonyan K., & Zhang B. (2017). The Kinetics Human Action Video Dataset. https://doi.org/10.48550/arXiv.1705.06950

[19] Kuehne H., Jhuang H., Garrote E., Poggio T., & Serre T. (2011). HMDB: A Large Video Database for Human Motion Recognition. In: *2011 International Conference on Computer Vision (ICCV 2011)*, pp. 2556-2563. https://doi.org/10.1109/ICCV.2011.6126543

[20] LeCun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W., & Jackel L. D. (1989). Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems*, 2, pp. 396-404. https://dl.acm.org/doi/10.5555/2969830.2969879

[21] Li T., Chan K., & Tjahjadi T. (2024). Variable Temporal Length Training for Action Recognition CNNs. *Sensors*, 24(11), 3403. https://doi.org/10.3390/s24113403

[22] Liao X., Cai Z., Chen J., & Liu T. (2023). Physics-based optical flow estimation under varying illumination conditions. *Signal Processing Image Communication*, 117(8), 117007. 10.1016/j.image.2023.117007

[23] Nguyen A., Yosinski J., & Clune J. (2016). Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. https://doi.org/10.48550/arXiv.1602.03616

[24] Petrogianni A., Koromilas P., & Giannakopoulos T. (2022). Film Shot Type Classification Based on Camera Movement Styles. In: Pinho A.J., Georgieva P., Teixeira L.F., Sánchez J.A. (eds) *Pattern Recognition and Image Analysis. IbPRIA 2022. Lecture Notes in Computer Science*, 13256. Springer. https://doi.org/10.1007/978-3-031-04881-4_48

[25] Petrogianni A., Koromilas P., & Giannakopoulos T. (2022). Dataset with classified film shots. GitHub. https://github.com/magcil/movie_shot_classification_dataset [1.09.2025]

[26] Plank B. (2022). The “Problem” of Human Label Variation: On Ground Truth in Data, Modeling and Evaluation. https://doi.org/10.48550/arXiv.2211.02570

[27] Prelinger Archives, https://publicdomainreview.org/collections/source/prelinger-archives [1.09.2025]

[28] Raj A. (2024). Understanding the [CLS] Token in BERT: A Comprehensive Guide. https://aditya007.medium.com/understanding-the-cls-token-in-bert-a-comprehensive-guide-a62b3b94a941 [1.09.2025]

[29] Rao A., Wang J., Xu L., Jiang X., Huang Q., Zhou B., & Lin D. (2020). A Unified Framework for Shot Type Classification Based on Subject Centric Lens. In: *Computer Vision – ECCV 2020: 16th European Conference*, Proceedings, Part XI, pp. 17-34. https://doi.org/10.1007/978-3-030-58621-8_2

[30] Risi S., & Togelius J. (2020). Increasing Generality in Machine Learning through Procedural Content Generation. *Nature Machine Intelligence*, 2(8). https://doi.org/10.48550/arXiv.1911.13071

[31] Savardi M., Kovács A. B., Signoroni A., & Benini S. (2021). CineScale: A dataset of cinematic shot scale in movies. *Data in Brief*, 36, 107002. https://doi.org/10.1016/j.dib.2021.107002

[32] Sevilla-Lara L., Liao Y., Güney F., Jampani V., Geiger A., & Black M. J. (2017). On the Integration of Optical Flow and Action Recognition. In: Brox T., Bruhn A., Fritz M. (eds) *Pattern Recognition. GCPR 2018. Lecture Notes in Computer Science()*, 11269. https://doi.org/10.48550/arXiv.1712.08416

[33] Simonyan K., & Zisserman A. (2014). Two-Stream Convolutional Networks for Action Recognition in Videos. In: *NIPS'14: Proceedings of the 28th International Conference on Neural Information Processing Systems*, 1, pp. 568-576. https://doi.org/10.48550/arXiv.1406.2199

[34] Singh A., Thounaojam D. M., & Chakraborty S. (2020). A novel automatic shot boundary detection algorithm: robust to illumination and motion effect. Signal Image and Video Processing, 14(4), pp. 645-653, (2020). https://doi.org/10.1007/S11760-019-01593-3

[35] Sonka M., Hlavac V., & Boyle R. (2015). Image processing, analysis, and machine vision. Cengage Learning.

[36] Soomro K., Zamir A. R., & Shah M. (2012). UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. https://doi.org/10.48550/arXiv.1212.0402

[37] Stroud J. C., Ross D. A., Sun C., Deng J., & Sukthankar R. (2019). D3D: Distilled 3D Networks for Video Action Recognition. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV 2020).* https://doi.org/10.48550/arXiv.1812.08249

[38] StudioBinder. (2020). Ultimate Guide to Camera Movement — Every Camera Movement Technique Explained [The Shot List Ep6]. YouTube. https://www.youtube.com/watch?v=IiyBo-qLDeM [1.09.2025]

[39] Sun D., Yang X., Liu M., & Kautz J. (2017). PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8934-8943. https://doi.org/10.48550/arXiv.1709.02371

[40] Sun Z., Ke Q., Rahmani H., Bennamoun M., Wang G., & Liu J. (2020). Human Action Recognition from Various Data Modalities: A Review. In: *IEEE Transactions*

on Pattern Analysis and Machine Intelligence, 45, pp. 3200-3225. https://doi.org/10.1109/TPAMI.2022.3183112

[41] Szeliski R. (2022). Computer Vision: Algorithms and Applications Second Edition. Springer.

[42] Tan M., Le Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, pp. 6105-6114. https://doi.org/10.48550/arXiv.1905.11946

[43] Tran D., Bourdev L., Fergus R., Torresani L., & Paluri M. (2015). Learning Spatiotemporal Features with 3D Convolutional Networks. In: 2015 IEEE International Conference on Computer Vision (ICCV 2015), pp. 4489-4497, https://doi.org/10.1109/ICCV.2015.510.

[44] Unity. Documentation for *NavMesh*, https://docs.unity3d.com/6000.2/Documentation/ScriptReference/AI.NavMesh.html [1.09.2025]

[45] Unity. Documentation for frustum culling AABB check https://docs.unity3d.com/6000.2/Documentation/ScriptReference/GeometryUtility.TestPlanesAABB.html [1.09.2025]

[46] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., & Polosukhin I. (2017). Attention Is All You Need. In: *31st Conference on Neural Information Processing Systems (NIPS 2017)*, pp. 6000-6010. https://doi.org/10.48550/arXiv.1706.03762

[47] Wan Z., Wang Z., Chung C., Wang Z. (2022). A Survey of Dataset Refinement for Problems in Computer Vision Datasets. *ACM Computing Surveys*, 56(7), 172, pp. 1-34. https://doi.org/10.48550/arXiv.2210.11717

[48] Wang P., Unal E., & Malhotra A. (2021). Timesformer implementation. GitHub. https://github.com/lucidrains/TimeSformer-pytorch [1.09.2025]

[49] Zendel O., Honauer K., Murschitz M., Humenberger M., & Dominguez G. F. (2017). Analyzing Computer Vision Data - The Good, the Bad and the Ugly. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, pp. 6670-6680. https://doi.org/10.1109/CVPR.2017.706

[50] Zhang C., & Berger C. (2023). Pedestrian Behavior Prediction Using Deep Learning Methods for Urban Scenarios: A Review. *IEEE Transactions on Intelligent Transportation Systems*, 99, pp. 1-23. https://doi.org/10.1109/TITS.2023.3281393

List of figures

List of tables

Summary

Camera Movement Classification Using a Deep Learning Model Trained on Synthetic Video Clips

Dynamic camera shot classification is a field that due to copyright and the significant data requirement of modern Deep Learning can benefit greatly from synthetic data generation. The thesis modeled camera movement in a way that allows it to be generated synthetically. For this purpose an open-source Unity application was designed and implemented, enabling customizable parameterization of generated video clips and scenes. The large potential of synthetic generators to introduce multi-modality of data is considered and might lead to the built application eventually being useful even outside of dynamic camera shot classification. The synthetic clip generator was used to generate a compact prototype of synthetic dynamic shot dataset. The choice of dataset generation parameters was discussed and both the parameters and generated data were evaluated using video Deep Learning methods like 3D convolutions and the Video Transformer. An approach to automated rating of synthetic data quality was proposed.