UPPSALA
UNIVERSITET

UPTEC STS 22028
Examensarbete 30 hp
Juli 2022

Labelling Customer Actions in an Autonomous Store Using Human Action Recognition

Oskar Areskog

Civilingenjörsprogrammet i system i teknik och samhälle

UPPSALA
UNIVERSITET

Labelling Customer Actions in an Autonomous Store Using Human Action Recognition

Oskar Areskog

Abstract

Automation is fundamentally changing many industries and retail is no exception. Moonshop is a South African venture trying to solve the problem of autonomous grocery stores using cameras and computer vision. This project is the continuation of a hackathon held to explore different methods for Human Action Recognition in Moonshop's stores. Throughout the project a pipeline for data processing has been developed and two types of Graph-Convolutional Networks, CTR-GCN and ST-GCN, have been implemented and evaluated on the data produced by this pipeline. The resulting scores aren't good enough to call it a success. However, this is not necessarily a fault of the models. Rather, there wasn't enough data to train on and the existing data was of varying to low quality. This makes it complicated to justly judge the models' performances. In the future, more resources should be spent on generating more and better data in order to really evaluate the feasibility of using Human Action Recognition and Graph-Convolutional Networks at Moonshop.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Abrie Cronje Ämnesgranskare: David J.T. Sumpter

Examinator: Elísabet Andrésdottir

Populärvetenskaplig sammanfattning

Automatisering har varit framtidens melodi ett tag nu i och med att fler och fler företag och industrier väljer att byta ut mänskliga resurser mot maskiner. Mat- och varuhandelsindustrin är inte ett undantag. Allt fler affärer byter ut bemannade kassor mot självscanningslösningar för att öka antalet kunder varje anställd kan hantera. Varför stanna vid självscanning? I takt med att AI utvecklas börjar en framtid skådas där den mänskliga faktorn vid en transaktion helt är obsolet.

Moonshop är ett sydafrikanskt företag som jobbar med att lösa autonom shopping på ett billigare sätt en de storskaliga lösningar som finns på marknaden idag. Målet är att bygga ett system för autonom shopping som endast använder ett fåtal filmkameror och på så sätt möjliggöra för implementering där vinstmarginalerna är lägre.

Det här projektet har tagit avstamp i ett hackathon Moonshop höll och har försökt utvärdera hur möjligt det är att kategorisera kunders handlingar i en butik med hjälp av ett par AI-modeller. Den största delen av projektet ägnades åt att bygga en pipeline för att genera de dataset som används vid träning av modellerna. Övrig tid har gått åt till att implementera, träna och utvärdera olika variationer av två modeller.

Modellerna som undersökts faller inom det fältet *människo-handlingsigenkänning* (*human action recognition*, egen översättning), eller *HAR*, och det mindre fältet *graf-konvolutionerande nätverk* (*graph convolutional networks*, egen översättning), eller *GCN*. HAR innefattar alla typer av maskininlärning och AI som försöker identifiera mänskliga handlingar och GCN är de modeller som gör det med grafen av ett mänskligt skelett som indata. GCN är en specialisering av den mer använda modelltypen *konvolutionerande neurala nätverk* (*convolutional neural networks*, egen översättning), eller *CNN*, och är alltså en typ av neurala nätverk.

Spatial Temporal Graph Convolutional Network, eller *ST-GCN*, är en modellarkitektur som introducerades 2018 och visade lovande resultat jämfört med dåvarande, konkurrerande modeller. Genom att behandla en sekvens av grafer som representerar ett mänskligt skelett i rörelse som en bild och skicka den genom en arkitektur liknande de i CNN-modeller gick det att med hög precision identifiera mänskliga handlingar. *Channel-wise Topology Refinement Graph Convolutional Network*, eller *CTR-GCN*, som introducerades 2021 bygger på och vidareutvecklar arkitekturen som presenterades med ST-GCN. Utvecklarna insåg att ST-GCN missar potentiellt relevant data genom att endast jämföra avstånd mellan noder som är bredvid varandra enligt skelettet. Genom att bygga in funktionalitet för att jämföra alla noder med varandra oavsett skelettförhållande lyckas de få ännu högre precision än ST-GCN. Ett exempel händerna som inte är bredvid varandra enligt skelettet men vars avstånd kan säga mycket om vilken handling som utförs.

Den råa data som fanns att tillgå existerade i form av JSON-data på en molndatabas och tre tillhörande videos på en annan. Varje JSON-video par representerar en kunds session i butik. Videorna visar kunden från olika vinklar i butiken och JSON-datan, som är handkategoriserad, innehåller information om vilka handlingar kunden utfört och vid vilka tillfällen. En handling är i det här fallet att antingen ta en vara från hyllan eller lägga tillbaks en. Pipelinen ingesterade denna data och matade ut ett samlat dataset med skelett-grafer på både handlingar och icke-handlingar. För att säkerställa att datan är fullständig görs en del kontroller under vägen. Det kontrolleras att videorna är synkade, att det inte är mer än en människa i bild under videons gång och att skeletten som hämtas håller någon form av kvalitet. På de segment där inga grafer lyckats utläsas placeras en graf med hjälp av extrapolering. Skeletten läses ur varje bild i filmen med hjälp av Googles pose-estimator MediaPipe Pose.

Modellerna tränades i olika variationer för att undersöka om hyperparametrarna har någon betydande påverkan. De hyperparametrar som experimenterades med är batch-storlek på indata och antalet grundkanaler i modellernas olika lager. Även effekten av att lägga graferna från alla tre videovinklar på varandra undersöktes. Det något underväldigande resultatet visar att ingen av modellerna presterar på en nivå där de kan implementeras problemfritt och effekten av att använda alla videovinklar på en gång är inte konsekvent nog att dra någon slutsats av. Det konstateras dock att detta inte behöver bero på modellerna utan att det lika väl kan vara en effekt av datasettet.

Datasettet lider många brister som inte har kunnat åtgärdas inom ramarna för det här projektet och det påverkar givetvis resultatet. Till att börja med var en stor del av den råa datan av för dålig kvalitet för att kunna användas ordentligt. Det ledde till att många datapunkter var tvungna att slängas och att många av de som användes är av bristande kvalitet. Ett annat problem var att det inte fanns någon tillräckligt bra pose-estimator för de rådande förutsättningarna att tillgå. Begränsningen till bara en person i bild åt gången reducerade antalet potentiella datapunkter ytterligare och de skelett som utlästes från bilderna var ofta kraftigt förvrängda. Till slut och delvis på grund av de ovanstående problemen blev datasettet för litet. Endast 1570 sekvenser av handlingar och icke-handlingar lyckades utläsas och det är för få för att säkert kunna träna en modell.

Med det sagt bör inte resultatet främst skyllas på dålig data. En mer uttömmande undersökning av den data som fanns att tillgå i början av projektet hade kunnat upptäcka problemen i förväg. Projektets mål hade då kunnat formuleras på ett sätt som varit mer realistiskt att uppnå. Detta är en viktig lärdom att ta med till framtida projekt. Frågan är om målet Moonshop önskar att uppnå ens är möjligt. Att försöka identifiera så komplexa handlingar med den begränsade data några videoströmmar kan ge är ett enormt svårt projekt. Det är möjligt att det aldrig kommer kunna lösas perfekt. Frågan blir då en fråga om affärer: går det att göra den tillräckligt bra för att handlaren ska kunna spara mer på implementation än vad hen förlorar på svinnet av dess ofullkomligheter?

Acknowledgements

This project and report is dedicated to Pieter Boon, a truly marvellous and generous man who were taken from us way too soon. Thank you for showing me Cape Town, Pieter!

I would like to thank everyone at Cape AI and Moonshop for the warm welcome, support and amazing memories you've all given me. Specifically I'd like to thank Byron, Abrie and Geoffrey for guiding me through this project.

I would also like to thank David J.T. Sumpter for always providing a sense of security and clear guidance during our meetings.

Contents

1 Introduction

Automation has been the story of the future for quite some time now as more and more industries and businesses change human resources for machines. Some people welcome this development with optimistic, open arms while others, often afraid to loose their work, watch it with growing unease and nostalgia for the days before. However, regardless of one's attitude towards this change the fact seem to remain that it's here to stay and will be hard to slow down or reverse. (Heller and Savargaonkar 2021)

Retail and grocery stores are not different to any of the other industries in this sense. Anecdotally it's been noticed that more and more stores adapt self-checkout systems in order to increase the amount of customers each personnel can manage. But why stop there? With better computers and development in machine learning, a future where the human element of shopping is completely eliminated becomes more and more feasible. (PYMNTS 2021)

Moonshop is a venture from the South African consultancy firm Cape AI that aims to solve the challenge of autonomous shopping in a cost-effective way. The goal is to develop a completely autonomous shopping system reliant only on cameras and machine learning. The possibilities such a system creates are many: for example, in a diverse country like South Africa it could mean being able to open many small stores close to poorer communities while in a country like Sweden, where labour is more expensive, it could mean providing a dying countryside with affordable and accessible grocery store options without exposing the vendor to too much financial risk.

In 2021, Moonshop held a hackathon in order to crowd source options for their Human Action Recognition systems. The contestants got a curated dataset of skeletal keypoints - keypoints with coordinates representing a human skeleton, see section 2.1.3 for further explanation - representing humans reaching into a shelf with groceries. The goal was to implement an AI model that labels whether the skeleton actually reaches for the shelf or not. (Zindi 2021) One of the winning submissions applied a model introduced in 2021 called Channel-wise Topology Refinement Graph Convolutional Network, or CTR-GCN (Chen et al. 2021). This model showed a lot of promise and did fit into Moonshop's already existing and preferred architecture.

This project builds upon the results of that hackathon. The aim is to further explore whether the CTR-GCN model is suitable for Moonshop using more realistic, less curated data; thus making it a more realistic scenario. In order to do this, a pipeline for data processing had to be developed. CTR-GCN and a more basic model named ST-GCN was then trained with and evaluated on the datasets generated from this pipeline. More than just training models, a hypothesis that it would improve performance to train the models on data compounded from several video feeds of the same action was also tested.

To specify, the goals of this project are:

- Develop a pipeline for data processing
- Train and evaluate CTR-GCN and ST-GCN models
- Determine if compounded datasets from several video feeds produce better models than datasets from single video feeds.

1.1 Limitations

Some limitations were applied in order to make the project doable. The biggest one were the choice to not distinguish between different types of actions (take or put) in the dataset. There are some motivations behind this decision.

Firstly, differentiating between them would increase the complexity a lot. It's not obvious even to a human eye whether a skeleton takes something from a shelf or puts something back without more information. There are AI models for extracting this information - eg. an object-in-hand estimator determining whether the hand holds something before and after the action - but finding and implementing these would've taken too much time. This was the intention at the beginning of the project but became unfeasible towards the end.

Secondly, the dataset wasn't big enough to divide into more classes. As will be evident in section 3, the biggest dataset extracted only held 1570 samples evenly distributed between actions and non-actions. Splitting the actions-half in two would further reduce the amount of data each model has to learn from.

Another limitation is how many variations will be explored. Due to time restraint this is a lower number than one could wish for. Because of this, the focus will be on the CTR-GCN model (introduced in section 2.2.1) with the big dataset (introduced in section 2.2.2) and only limited tests will be arranged for the other model and dataset.

1.2 Essay Layout

Section 2.1 details the data processing pipeline and the datasets created from it. Section 2.2 goes into details about the models being tested, how they are trained and then evaluated. Section 3 will then show some examples of the data together with evaluation scores for all trained models and test scores for some chosen models. Section 4 will reason around some of the difficulties and problems and draw conclusions based on the results. Finally, Appendix A is a short dictionary explaining some of the terms introduced in the report and Appendix B is a collection of the graphs related to section 3.3 and 3.2.

All code referenced to in this paper can be accessed (with the right credentials) on GitHub. [1]

2 Methods

This section will detail the different methods and techniques applied in the project. Subsection 2.1 will detail the methods used in the data preparation stage; the stage that preoccupied most of the time of the project. Each step from data gathering to final formatting will be covered. Subsection 2.2 will introduce the model architectures used and detail the training and evaluation methods applied.

All development was done on MacOS. The code is written in Python 3.X using either Microsoft Visual Studio Code or Google Colab.

2.1 Data Preparation

This section will detail the different methods and steps used it the data preparation step. Section 2.1.1 describes the shape of the raw data, how to access it and some pre-processing done before gathering it. Section 2.1.2 introduces some problems with the raw data and the methods applied to address them. Section 2.1.3 will go into details of the data relevant to extract and how it its extracted while section 2.1.4 introduces the way in which the refined data is stored. Finally section 2.1.5 will describe how the final presentation of the data before using it should be formatted and how it is stored.

2.1.1 Raw Data, Pre-processing and Gathering

Moonshop had three operating stores before the project start so there was a database of historic, labelled data to collect. However, the data is split and stored in a way that makes it incompatible for the intended use case; training *Graph-Convolutional Neural Networks*. Therefore, this data is considered *raw*.

The raw data stored by Moonshop details customer's visits in the stores. Each customer's visit is called a *basket*. Thus, the data is stored as unique *baskets*. When a customer enters a store, a new basket with a unique *basket-id* is created. Data about the basket is gathered until the customer leaves the store. Each basket is stored as two separate parts on two separate databases: videos are stored on Amazon Web Services, or *AWS*, and a dictionary of data is stored on Google's *Firebase*. These are linked via the basket-id. In both cases an id unique to the *store* (the physical location) and one unique to the *vendor* (the owner of the store) is stored with the data as well. See figure 1 reference.

[1] https://github.com/arescout/MastersProject

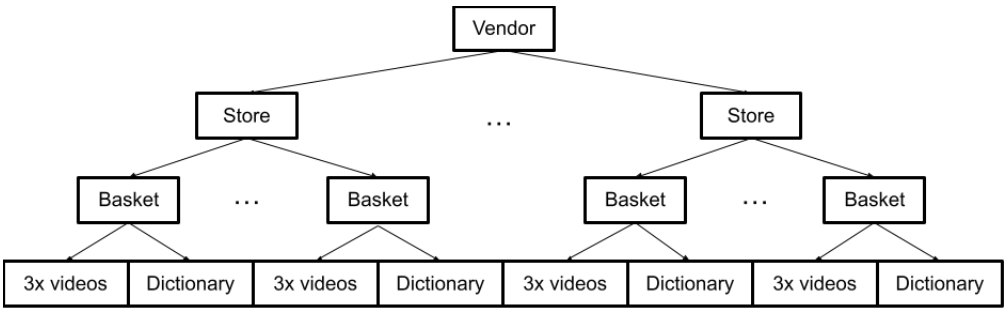

Figure 1: Structural relation of the data

Each basket has three videos connected to it. These videos show the same sequence from different angles. What angles are showed depends on which store the footage is taken in but in all cases the cameras were positioned in a corner of the store's room, looking down on the customer. The dictionary related to the basket holds a lot of information relevant for different parts of operations. Relevant for this project is start and stop timestamps as well as timestamps for each *action* present in the basket. An action is a customer either taking something from a shelf or putting something back on the shelf. As mentioned in sec 1.1, this project will not distinguish between the different types of actions. The actions are manually labelled during the running of the store.

The data is gathered by first deciding which baskets to download, or *pre-processing the baskets*. This is done by accessing the database of dictionaries and fetching all with more than zero actions. Here, it's possible to decide which stores one wishes to look at and could, for example, leave one out if the data from that store is deemed too unreliable to use. After they're fetched, the start time of each basket is compared to the stop time of every other basket in the same store. This is done in order to remove any overlapping baskets. The reasons for removing overlapping baskets will be discussed in section 2.1.2. The selection of baskets resulting from the process described is then saved for future access. This is done by saving all the basket-ids locally to a text-file.

Downloading baskets are done separately from the pre-processing. The download script takes any text-file with basket-ids as input, connects to AWS and Firebase and downloads the dictionary and all three videos. These are then stored locally to different folders and identified by being named the basket-id (and camera number if it's a video, the numbers being 1, 2 or 3).

2.1.2 Cleaning

The raw data has two main issues that needs to be sorted out before extracting the relevant features from it. The first issue stems from faults in the dataset while the second stems

from a limitation in the applied techniques.

First, it is not guaranteed that all three videos are in sync with each other. This is because the videos from the stores can be distorted due to connectivity issues during operations. The distortion affects both video synchronization and footage quality. This issue is solved by comparing the number of frames extracted from each video (see section 2.1.3 for a description of the extraction), i.e. the length of each, and making sure that they are all within one second of each other. While this comparison being true is not a guarantee of the videos being synced it is assumed that if they are about the same length they should show the same thing. Other techniques for assuring sync were explored - e.g. a text-reader model was implemented to read and compare the timestamp visible in each video - but the distortion was simply too bad to get any reliable way of comparing videos out of these. In many of the cases the distortion made the timestamp unreadable and the problems with synchronization affected the timestamps as well in an unpredictable way, rendering them unusable even if they were reliably readable.

Second, the pose estimator - the model used for extracting keypoints - used in section 2.1.3 only works with one person in frame. This is the reason for checking for any overlapping baskets as described in section 2.1.1. This issue is addressed in two ways. The first is obscuring any non-essential fields of the frames. In some instances there are people walking by outside of the store that are picked up by the camera. When processing each frame, a uni-coloured block is therefore painted over these fields, eliminating this risk. The second way is by processing each extracted action frame and non-action frame (see section 2.1.3 for a description of action frame) through a person detection model. This is done after the frames are obscured. The one used is called *Yolov5s*[2] and returns a list of labelled, cropped bounding boxes with a confidence score related to each bounding box. Trial and error resulted in a detection confidence threshold of 0.50. If the model labels a frame as having more than one person the basket is discarded. The same happens if no frame in that basket is labelled as having any people at all. This can happen as an effect of the distortion described earlier in this section.

2.1.3 Extracting

As will be described in section 2.2, *Graph-Convolutional Neural Networks* takes sequences of skeletal joint *keypoints* as inputs. Thus, the aim of the extraction part is to go from a movie-file to any sort of sequential datatype (i.e. `array`, `ndarray` etc.) populated with sets of keypoints. This is done in four steps: *reading*, *splitting*, *extracting* and *interpolating*. Figure 2 gives an overview of the process.

Firstly, the raw data has to be read. The dictionary is read using the built-in `json`[3]

[2]https://pytorch.org/hub/ultralytics_yolov5/; visited 2022-06-07
[3]https://docs.python.org/3/library/json.html; visited 2022-06-07

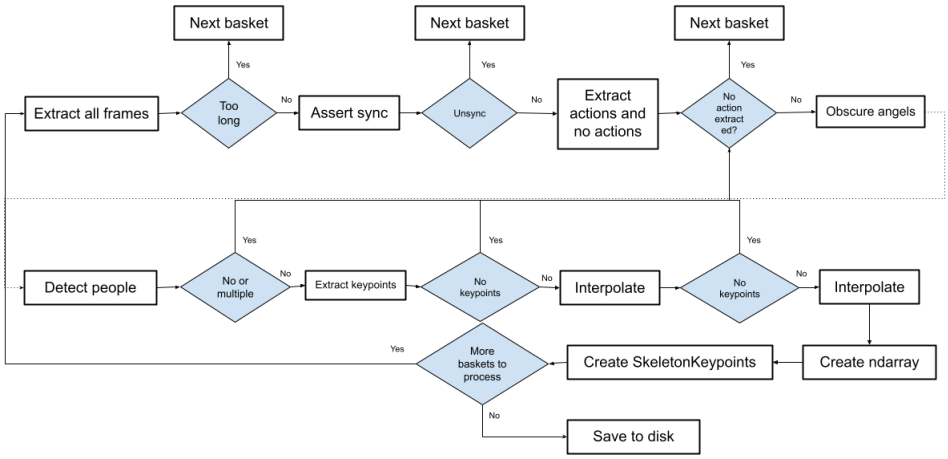

Figure 2: The data processing pipeline

library and stored in RAM. Each movie-file is then read and turned into `ndarrays` with frames of RGB-values using the `openCV`[4] library. Each `ndarray` of frames is then stored in one single `array`. When reading the video-files an upper limit of frames read is implemented in order to avoid RAM-crashes. This is a risk due to inconsistencies in the raw data, with some videos being very long. A rather arbitrary limit of 2000 frames, or 133,33 seconds, was chosen. This is longer than most basket sessions and should thus capture all real samples. If the limit is reached, the basket is discarded in its entirety.

Secondly, the read data has to be split into sections centered around the interesting parts - the actions present in the basket dictionary. For x actions present, each action's labelled frame is calculated based on the labelled timestamp. Before and after each labelled action frame, a padding of p seconds ($p = 2$ in my implementation, decided on after some trial-and-error of trying to get as much of the action as possible while minimizing frames before and after) is extracted and stored as new `arrays`. The number of frames extracted per second of padding depends on the frames per seconds, or FPS, of the footage ($FPS = 15$ in the case of all cameras and videos present in this dataset). That means that for each basket, a set of x `arrays` with $2p * FPS$ action frames is extracted.

However, training a model on single labelled data isn't very effective; *non-action* frames have to be extracted as well. For each action mentioned above an equally long set of frames without any action is extracted. The frames taken are the $2p * FPS$ frames just before the first correlating action frame. A check is done to make sure that these frames aren't part of any other set of action frames. If that's the case, they are simply dropped and not replaced. Thus, for each basket a set of $[x, 2x)$ `arrays` with $2p * FPS$ frames

[4]https://opencv.org/; visited 2022-06-07

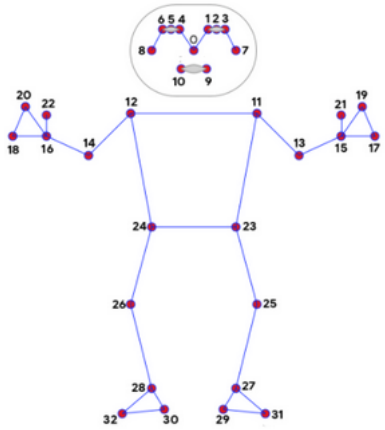

Figure 3: The skeleton generated by MediaPipe Pose (MediaPipe 2022)

per camera angle is extracted and stored in RAM.

Thirdly, the skeletal joint keypoints have to be extracted. For this, the Google developed `MediaPipe Pose`[5] library is used. It contains a *pose estimator* that takes frames as input and outputs an `ndarray` of 33 keypoints with a x-coordinate, an y-coordinate and a confidence score for each. See figure 3 for an example of the skeleton extracted. Each set of action and non-action frames is processed through the estimator and returned an `ndarray` with the shape `(number of frames,33,3)`. After this, a check is done to make sure that the result actually contains any keypoint coordinates and not just a list of zeros, which if the result of not reaching the confidence threshold. If this is the case, that action or non-action is dropped.

Finally, it has to be made sure that each sequence contains a continuous set of keypoints. This is not obvious since the pose estimator will return a set of zeros whenever the confidence score fails to reach the confidence threshold (set to 0.70 after trial-and-error). In order to fill any potential gaps, the `interpolator`[6] method in the `scipy`[7] library is used. This will interpolate any missing points in a linear fashion based on the points before and after. If no keypoints are available in the entire sequence it is dropped since this would result in very unreliable interpolations. Obviously this isn't a perfect method since a skeleton's movements seldom are linear but it's a fairly quick fix to a complicated problem. More research could be done into the field of skeleton keypoints interpolation but that could probably be a project on its own. The last thing that happens is that the `ndarrays` of keypoints from each camera view are compounded into one single `ndarray`, this one having the shape `(9, number of frames, 33)` with

[5]https://google.github.io/mediapipe/solutions/pose; visited 2022-06-07
[6]https://docs.scipy.org/doc/scipy/tutorial/interpolate.html; visited 2022-06-07
[7]https://docs.scipy.org/doc/scipy/index.html; visited 2022-06-07

9 being the three sets of keypoints per joint.

2.1.4 Storing

At this point, all information needed to train a Graph-Convolutional Neural Network is extracted and stored in RAM. However, it is not efficient to perform this process every time a model is to be trained (seeing how it took about 20 hours on a 2020 Macbook Pro) and it could useful to keep information other than the data ingested into the models, like dataset-id and basket-id, for future reference. Because of this, a custom class - `SkeletonKeypoints` - was created to organize and store all required information. This section will explain the use of the different variables and methods defined in the class.

One instance of `SkeletonKeypoints` is created for each action or non-action extracted by the methods explained in 2.1.3. `SkeletonKeypoints` holds the following variables:

- `keypoints`: The `ndarray` of compounded keypoints on dimension `(9, number of frames,33)`.
- `n_o_keypoints`: The amount of joints per frame.
- `n_o_frames`: The number of frames in the sequence.
- `n_o_cams`: The number of camera angles in the dataset.
- `split_keypoints`: An `array` with `ndarrays` with separated keypoints.
- `timestamp`: When creating a new dataset, a timestamp is generated. This timestamp acts as a unique identifier for that dataset. It is stored here in order to reference the files and directories created for that dataset.
- `label`: The binary label of the instance with 0 = no action, 1 = action.
- `labels`: A list with named labels in the place corresponding to that label's number (i.e. `["non_action","action"]`.
- `data_dir`: The relative location to the directory where data generated by the process is stored.
- `frames_path`: The relative location to the directory where frames saved by the process is stored. Each instance has one video per camera view related to it in this folder.
- `action_numb`: What number in the sequence of actions in that video this action has. E.g. if the basket holds four actions and this is the third one extracted, `action_numb` would be **3**.

- `tensor_pair`: An `array` containing two `tensors`: one created with `keypoints` and one containing `label`.
- `basket_id`: The basket-id of the video this instance is extracted from
- `video_paths`: An `array` containing the relative paths to the created videos mentioned at `frames_path`.

Following these variables are a set of methods to perform different tasks. This report won't go into them in detail but some notable functionalities to mention are: a method for receiving a frame from all camera angles and their associated keypoints, one method to get and annotate a frame with its keypoints, annotate and save the entire instance to disk, make `tensor_pairs` from the split keypoints and combine two datasets of `SkeletonKeypoints` with different `timestamps` with each other.

When all baskets in a dataset are fully processed the result is an `array` populated with instances of `SkeletonKeypoint`. In order to access this list in the future it is stored to disk as a binary using the built-in library `pickle`[8].

2.1.5 Formatting

In order to use the data extracted in section 2.1.3 it has to be formatted in the specific way the models expect. As will be described in section 2.2.1, the models for this project are implemented using `PythorchLightning`[9] which expects `tensors`[10] or `DataLoaders`[11] as input. `DataLoader` is a datatype defined in the `torch`[12] library facilitating storing and handling sets of `tensors` in an accessible way. It stores the data in *batches* ready to insert into a model. The *batch size* is defined upon creation.

In this case, a dataset consisting of `SkeletonKeypoints` instances is loaded from disc using `pickle`. A new `array` is then created and populated with all the instances of `tensor_pairs` present in the `array`. This new `array` is then shuffled using `Python's` built in `random`[13] library and split into training, validation and test arrays based on pre-defined fractions. The `seed` is globally set to 1234 throughout the project. The actual splits in this project are presented in table 1. They were calculated as testing being 10% of the entire set and validation being 20% of the remaining set. This way, enough data is separated to test upon while ensuring that the validation set, used in the training, is bigger. When the train, validate and test `arrays` are created they are turned into `DataLoaders` with a given batch size and saved to disc using the

[8]https://docs.python.org/3/library/pickle.html; visited 2022-06-07
[9]https://www.pytorchlightning.ai/; visited 2022-06-07
[10]https://pytorch.org/docs/stable/tensors.html; visited 2022-06-07
[11]https://pytorch.org/tutorials/beginner/basics/data_tutorial.html; visited 2022-06-07
[12]https://pytorch.org/; visited 2022-06-07
[13]https://docs.python.org/3/library/random.html; visited 2022-06-07

Set	Fraction
Training	72 %
Validation	18 %
Testing	10 %

Table 1: Fractions for the different splits in the dataset

`torch.save` method. Each `DataLoader` now contains batches of batch size many `tensor`-pairs with the shapes `tensor.size(9,number of frames, 33)` and `tensor.size(1)`.

2.2 Model Training

This section will describe the models used to detect customer actions. Section 2.2.1 will detail the models and their architecture, section 2.2.2 will describe their implementations and section 2.2.3 will introduce the methods used when evaluating them.

2.2.1 Models

As mentioned in the introduction there had been some exploration of this topic before this project started. That did not mean that the models tested necessarily were the optimal choices. In order to make sure that the models used held weight an initial research into *Human Action Recognition*, or *HAR*, was done. HAR is a field within Artificial Intelligence that focuses on accurately label human actions using a plethora of different types of methods and models (Al-Faris et al. 2020, p. 1). However, the outcome of this research resulted in two fitting models of which one happened to be the model implemented in the hackathon. The two models are both versions of *Graph-Convolutional Neural Networks*, or *GCN*.

A GCN is a version of a *Convolutional Neural Network*, or a *CNN*. A CNN is a type of *neural network* most commonly used for classification or computer vision tasks. A CNN mainly utilizes *convolutional layers*, *pooling layers* and *fully connected layers* to extract features in images. Each layer of *convolutional* and *pooling layers* increases the complexity of the features recognized. The input to a CNN, when using it for image recognition, is usually a matrix of pixels in 2 or 3 dimensions containing the RGB-values of the image. (Education 2020)

Just like a CNN, a GCN uses layers of *convolutions*, *poolings* and *fully connected* to extract features. The difference lies in the input. Instead of extracting features from a matrix of pixels, a GCN will extract them from a matrix of node coordinates and an adjacency matrix. (Casalegno 2021) Seeing how Moonshop prefer to

use skeletal keypoints rather than the actual image, the GCN approach is more fitting the a CNN approach. The adjacency matrix is the same for all input images from the same type of graph and is therefore part of the model. The input images are instances of `ndarrays` with the dimensions `(number of features, number of frames, number of nodes)`. `Number of frames` depends on the length of the image sequence used, `number of nodes` depends on what type of graph is used and `number of features` depends on the dimensionality of the graph.

ST-GCN

The *Spatial Temporal Graph Convolutional Network*, or the *ST-GCN*, was introduced in 2018 by researchers at The Chinese University of Hong Kong. It is, according to their paper, the first attempt to apply a GCN on a graph of skeletal keypoints over both *space* (spatial) and *time* (temporal). Its aim is to extract features from both joints' positions relative to each other in space as well as relative to their own in time to classify human actions. (Yan et al. 2018, p. 1)

ST-GCN defines an undirected skeletal temporal graph $G = (E, V)$. Here, the node set $V = \{v_{ti} | t = 1, ..., T, i = 1, ..., N\}$, where T is the number of frames and N is the number of joints, holds all skeletal nodes extracted from a video sequence. $E = \{E_S, E_F\}$ is a set of two subsets, each containing a set of edges. $E_S = \{v_{ti}v_{ti} | (i, j) \in H \}$, where H is the set of bone connections in the particular skeletal pose estimator used, represents the natural edges between the nodes in each frame and $E_F = \{v_{ti}v_{(t+1)i}\}$ contains the connection between each specific node between the frames. (Yan et al. 2018, p. 3)

Furthermore, it wouldn't really make sense to compare all joints distance to all other joints since that's not how the human body works. For example, a foot joint's distance to it's knee should contain more information about an action than, say, that foot's distance to the head. A number of partition strategies are suggested to solve this. The strategies describe different ways to define what nodes to compare to each other. The strategy relevant for this project is the *spatial configuration partitioning*. It divides the neighbour set into three subsets: 1) the root node, 2) the neighbouring nodes that are closer than the root node to gravity center of the skeleton and 3) the rest of the neighbouring nodes. See figure 4 for a visual reference. (Yan et al. 2018, p. 5)

Finally, the ST-GCN model is composed of 9 layers of ST-GCN units, or *spatial temporal graph convolution operators*. These are layers designed to extract features from a spatial temporal graph. Each unit has a number of input- and output channels related to a variable named `base_channels`, or BS. In the paper and base model, we have that $BS = 64$. The first three layers have $1 * BS$ output channels, the following three layers have $2 * BS$ output channels and the final three have $4 * BS$ output channels. The size

Figure 4: An example of the spatial partitioning (Yan et al. 2018, p. 5)

of *BS* will determine how many details each convolution will extract. Each layer has a kernel size of 9 and the 4th and 7th layers have stride 2 instead of 1 for pooling. After this, a global pooling layer is applied to reduce the resulting tensor to a 256 dimension feature vector. This vector is then fed into a *SoftMax* classifier. *Stochastic gradient descent* is applied in the learning process with a *learning rate* of 0.01. This is reduced by 0.1 per every 10th epoch. (Yan et al. 2018, p.6)

CTR-GCN

Building upon the work done with the ST-GCN model, researchers at the Chinese Academy of Sciences realized an inherent flaw in the way topology and partitioning was handled. All methods proposed by Yan et al. 2018 limited their topology to neighbouring nodes as defined by the given skeleton. While these relations are important, they risk missing out on features not presented by the bones of the body. For example, while one hand's distance to its elbow is an important feature, that hand's distance to the head might be equally or more important when trying to identify an action. (Chen et al. 2021, p.1)

The *Channel-wise Topology Refinement Graph Convolutional Network*, or *CTR-GCN*, propose a method to address this flaw. The CTR-GCN does this by taking high-level features transformed from input features and then dynamically inferring topologies in order to capture the features and correlations between joints under different kinds of motion features. The dynamic features are inferred channel-wise and features are aggregated channel-wise with corresponding topology to get the final output. This is done in three steps: feature transformation, channel-wise topology modelling and channel-wise aggregation. (Chen et al. 2021, p.3)

The CTR-GCN model is built up of *"basic blocks"*. Each *basic block* consists of a *spatial modelling* module, a *temporal modelling* module and a final *ReLu* function. The *spatial modelling* consists of three parallel CTR-GC units whose outputs are added and sent into a *batch normalization* and *ReLu* module. The *temporal modelling* module consists of

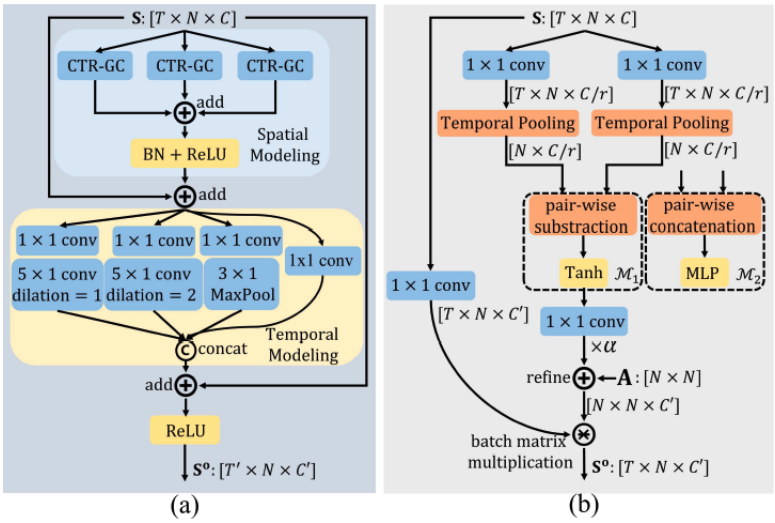

Figure 5: (a) A visualization of the basic block. (b) A visualization of the CTR-GC unit. $T =$ the number of frames per input, $N =$ number of keypoints per frame and $C =$ number of features per keypoint. (Chen et al. 2021, p.6)

four branches doing different combinations of *1x1 convolutions*, *5x1 convolutions* and *3x1 max poolings*. The output of the four branches are concatenated. See figure 5 for visual representations of the two systems. (Chen et al. 2021, p.6)

The model uses `base_channels`, or *BS* much in the same way as the ST-GCN model described above does. In total, the network lines ten *basic blocks* with a final *global average pooling* and a *SoftMax classifier*. The first four blocks have $1 * BS$ channels, the following three have $2 * BS$ channels and the final three have $4 * BS$ channels. The 5th and 8th block halves the *temporal dimension* by *strided temporal convolutions*. *Stochastic gradient descent* is used when training with momentum 0.9 and weight decay 0.0004. (Chen et al. 2021, p.6)

2.2.2 Training

The implementations of the *ST-GCN* and the *CTR-GCN* models are practically identical. This section will therefore mostly describe the implementation of the *"model"*, implying that this represents both models. If that's not the case it will be explicitly expressed at that specific point.

The models are available in their original, extended forms within publicly available repos. However, it's not these exact versions that are implemented in this project. This is mainly since those versions are built for the experimental settings described in their papers and not easily translated to the specifics of this scenario. Also, some changes to

the models had to be made which was easier to do in the controlled setting of a familiar code base.

The classes and methods related to the model architecture are basically copied in their entirety. These classes include the main `Model` class holding the architecture and support methods, the `Graph` class defining the adjacency matrix and partitioning strategies, and several `layer` classes defining, for example, the *CTR-GC* unit. While the latter two classes are left mostly untouched, the `Model` class has been altered. The most obvious change is that it is implemented using the `pytorch_lightning` library. It is a `Python` library building upon `PyTorch`, automating most of the architecture around a model such as training and validation. Crucially it also logs the training and saves the latest version of the model automatically, together with measurements from the training.

Other than implementing `pytorch_lightning`, some minor changes was made to the model. The `base_channels` variable was introduced in order to facilitate experimentation with that hyper-parameter. The number of classes, input channels, number of frames, number of joints etc. was made dynamic as well, in order to fit the parameters to this specific data. The possibility to turn off GPU-utilization by making `cuda`, a GPU driver, a boolean was done in order to run the code on a machine without GPU-support. Finally, the model and graph was changed to fit the pose estimations created by MediaPipe.

The models are defined within the project repo but in order to access a GPU and speed up the training, the actual training loops are written and ran in *Google Colab*; a service for hosting *Jupyter Notebooks* on *Google Drive* and run them using freely available GPUs. The loop consists of loading the dataloaders with the correct batch size and camera setup, using them to train a model set up with the same parameters and a defined base channel size, and afterwards manually moving and renaming the saved, trained model. All versions of trained models are trained from the ground up, no transfer learning has been implemented.

Several variations of the two models have been trained in order to find the optimal combination. Mostly it's the `base_channels` variable that's been varied but the batch size has also been experimented with. For every version, four sets of the data were trained on: a set with the keypoints from all three angles combined (or the *compounded* dataset) and one set per camera angle. They are named 0, 1, 2, 3 and 4 respectively, with 0 being the compounded set. Finally, a reduced dataset with less noisy data, due to the data from one store being very distorted, was generated and experimented with. The bigger dataset with 1570 samples is named *D1* and the smaller with 436 is named *D2*. All variations are presented in figures 10, 11 and 9.

2.2.3 Evaluation

The evaluation of the models looks the same regardless of that model's variation. It is handled by a custom function, `evalModel`, and produces a host of measurements relevant to consider when evaluating. The method can be used with both validation and test sets.

First, the method produces predictions based on the data and model given. The predictions are decimals in the range $0 - 1$, representing the probability that the sample is an action. These predictions are then used to find the *optimal threshold* for this label. This is done by calculating the *F1 score*, using the `f1_score`[14] library available in `sklearn.metrics`[15], for each threshold value in the range $0 - 1$ with a 0.001 step.

The F1 score is a metric combining *Precision* - the fraction of true positives found of all found positives - and *Recall* - the fraction of true positives found of all true positives - and works well on both skewed and evenly distributed data. It is the mean of the two measurements, defined as

$$F1\_score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{1}$$

and thus gives equal weight to both measurements. The metric is a good alternative for approaching the precision-recall trade-off. The optimal threshold is then decided by taking the threshold scoring the biggest F1 score. (Korstanje 2021)

When the optimal threshold is decided, that threshold is used to calculate the *accuracy* of the model. This is done by comparing the predictions generated earlier to the threshold and labelling any prediction greater than the threshold an action. Each labelled prediction is then compared to its true label, present in the data set, and the number of accurate labels are counted towards a score. The accuracy is then defined as the score's fraction of all samples in the dataset, converted to percentage (Korstanje 2021).

Following the accuracy, the *AUC Score* is calculated and a *ROC Curve* is created. These actions are done using the `roc_auc_score`[16] and `RocCurveDisplay`[17] methods respectively, both present in the `sklearn.metrics` library. The ROC Curve is generated by plotting the *True Positive Rate*, or *TPR*, on the y-axis against the *False Positive Rate*, or *FPR*, on the x-axis. The AUC Score is the area under the ROC Curve. The score is a measurement of how well a model labels samples correctly: the higher

[14]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html; visited 2022-06-07

[15]https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics; visited 2022-06-07

[16]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html; visited 2022-06-07

[17]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html; visited 2022-06-07

the score, the better the model. A score of 50% means that the model is equally likely to label a measurement true or false regardless of its true state. (Narkhede 2018)

Finally, a *confusion matrix* is generated. This is done using the `confusion_matrix`[18] method available in the `sklearn.metrics` library. This method only generates the scores of the matrix and in order to plot it, the `ConfusionMatrixDisplay`[19] method present in the same library can be applied. Using the confusion matrix, the TPR is calculated one final time, manually this time.

3 Results

The following sections will present the results of the methods described in section 2. Section 3.1 will show some examples of data representation while section 3.2 will detail validation results for the different models trained and 3.3 will present testing results for the models with the most promising validation scores.

3.1 Data Preparation

The data preparation was the biggest part of the project time-wise. The result is an almost automated process that combines, prunes and refines two different sources of data into one, instantly usable database. The process should also be generic enough to instantly apply to new data being ingested to the raw databases.

Following, figure 6 is one example of a basket JSON-file with all but the relevant fields removed. Figure 7 is an example of the frames from one basket video before any processing is done. Note that, as described in section 2.1.1, the video is actually three separate videos, here combined into one figure to make the presentation easier. The faces in all frames, except for one frame in figure 8 in order to show the annotated face, have been obscured specifically for this paper in order to not reveal anyone's identity and is not representative of their state when in the process.

[18] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html; visited 2022-06-07

[19] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html; visited 2022-06-07

Figure 7: Example of a frame from an unprocessed basket video


```
{
    "shoppingEvents": [
        {
            "action": "take",
            "id": "e3cK0pyFYv7Rtbcg8U2z",
            "name": "Nestl\u00e9 KitKat Chunky Chocolate Bar 40g",
            "time": "2021-10-07 13:23:22.089000+00:00"
        },
        {
            "action": "take",
            "id": "hrgIoOqvavIbIrtg97Wn",
            "name": "Willards Big Korn Bites Barbecue Flavoured Maize Chips 50g",
            "time": "2021-10-07 13:23:27.089000+00:00"
        }
    ],
    "store": "YluoXEkaZvmFsr5d43Jj",
    "timeEnd": "2021-10-07 13:23:46.089000+00:00",
    "timeStart": "2021-10-07 13:23:11.089999+00:00",
    "vendor": "checkers"
}
```


Figure 6: Example of a basket JSON-file with the only the fields relevant for the data preparation process

Finally, figure 8 show the same frames after they've been processed. Notice the annotated skeleton described in section 2.1.3 and the obscured angels described in section 2.1.2. The annotations on those frames are representative of the content of the `SkeletonKeypoint` instance. It's interesting that the frame is the exact frame labelled in the basket as the action. However, it looks like the action occured some second earlier. This is a good example of noise in the data and the importance of adding enough padding as discussed in section 2.1.3. In this case the noise is probably due to sloppy labelling and not distortion.

The biggest dataset generated, D1, have 1570 samples in it. It is generated from 1261 baskets and took around 20 hours to generate on a modern CPU. The splits and fractions of this dataset is presented in table 2. The biggest time-sinks were the object detector described in section 2.1.2 and pose estimator described in section 2.1.3. However, these

Figure 8: Example of a frame from a processed basket

Set	Actions	Non-Actions	Fraction
Total	50 %	50 %	100 %
Train	49 %	51 %	72 %
Validate	55 %	45 %	18 %
Test	46 %	54 %	10 %

Table 2: Splits and fractions in the dataset of size 1570 samples

two parts are paramount to the process and can't be avoided. They would probably work quicker on a GPU though. The lack of access to a local GPU and the limit on how much data one can upload to a remote solution like Google Colab necessitated the use of a CPU. The smaller dataset, D2, with less noisy data has 436 samples.

3.2 Training

The training was a time-consuming and, at times, irritating task. One big obstacle was the lack of access to a reliable GPU. Google Colab, which the training took place on, has a seemingly arbitrary time limit and could at any time cancel a process mid-training. Much of the day it would lock one out for having used the GPU too much the day before.

However, despite this obstacle a number of models were trained. The validation scores for the trained ST-GCN models are displayed in table 3. For this table, and all other tables showing model scores, the **ID** field is made up of *batch size* - *base channels* - *model version*, **OT** is short for optimal threshold and **F1** is the F1-score of that threshold. The highest score in **AUC**, **Accuracy** and **TPR** are presented in bold (except for **TPR** where some models scored perfectly which is unreasonable, since that could as well be a model labelling all samples as true). Table 4, 5 and 6 shows different variations of the CTR-GCN model's validation scores. The confusion matrix and ROC curve for each model is provided in Appendix B.

ID	OT	F1	AUC	Accuracy	TPR
16-32-0	0.142	0.778	0.746	0.716	0.903
16-32-1	0.223	0.716	0.756	0.645	0.920
16-32-2	0.172	0.765	0.786	0.699	0.952
16-32-3	0.133	0.724	0.734	0.645	0.956
16-64-0	0.035	0.772	0.759	0.709	0.897
16-64-1	0.023	0.693	0.728	0.585	0.964
16-64-2	0.159	0.757	0.763	0.702	0.903
16-64-3	0.263	0.709	0.759	0.688	0.781
64-64-0	0.035	0.790	0.722	0.727	**0.935**
64-64-1	0.106	0.749	0.780	0.695	0.934
64-64-2	0.012	0.753	0.766	0.667	0.986
64-64-3	0.252	0.757	**0.799**	**0.741**	0.832

Table 3: ST-GCN with D1 dataset Validation Scores

ID	OT	F1	AUC	Accuracy	TPR
16-16-0	0.000	0.709	0.482	0.550	1.000
16-16-1	0.327	0.668	0.624	0.535	0.964
16-16-2	0.000	0.679	0.543	0.514	1.000
16-16-3	0.448	0.687	0.632	0.571	0.971
16-32-0	0.328	0.723	0.664	0.624	0.890
16-32-1	0.243	0.655	0.595	0.493	**0.993**
16-32-2	0.352	0.700	0.555	0.578	0.959
16-32-3	0.174	0.677	0.629	0.535	1.000
16-64-0	0.153	0.771	**0.685**	**0.699**	0.923
16-64-1	0.000	0.654	0.528	0.486	1.000
16-64-2	0.011	0.679	0.476	0.518	**0.993**
16-64-3	0.294	0.689	0.674	0.599	0.912
16-128-0	0.005	0.721	0.629	0.585	0.974
16-128-1	0.166	0.693	0.648	0.606	0.912
16-128-2	0.268	0.703	0.659	0.589	0.945
16-128-3	0.268	0.706	0.671	0.649	0.869

Table 4: CTR-GCN with D1 dataset, Batch Size = 16 Validation Scores

ID	OT	F1	AUC	Accuracy	TPR
4-64-0	0.232	0.717	0.540	0.578	0.974
4-64-1	0.000	0.654	0.430	0.486	1.000
4-64-2	0.000	0.679	0.545	0.514	1.000
4-64-3	0.064	0.667	0.543	0.521	0.985
16-64-0	0.153	0.771	0.685	**0.699**	0.923
16-64-1	0.000	0.654	0.528	0.486	1.000
16-64-2	0.011	0.679	0.476	0.518	**0.993**
16-64-3	0.294	0.689	0.674	0.599	0.912
32-64-0	0.209	0.741	0.677	0.670	0.858
32-64-1	0.297	0.678	0.674	0.610	0.847
32-64-2	0.244	0.689	0.641	0.589	0.924
32-64-3	0.092	0.680	0.627	0.543	1.000
64-64-0	0.059	0.739	0.691	0.624	0.968
64-64-1	0.382	0.701	**0.708**	0.674	0.788
64-64-2	0.188	0.725	0.627	0.621	0.972
64-64-3	0.090	0.694	0.671	0.574	**0.993**

Table 5: CTR-GCN with D1 dataset, Base Channels = 64 Validation Scores

ID	OT	F1	AUC	Accuracy	TPR
16-64-0	0.014	0.735	0.533	0.603	0.977
16-64-1	0.021	0.780	**0.600**	**0.654**	**0.980**
16-64-2	0.021	0.696	0.489	0.551	1.000
16-64-3	0.019	0.729	0.592	0.590	0.977

Table 6: CTR-GCN with D2 dataset Validation Scores

Model	Dataset	ID	OT	F1	AUC	Accuracy	TPR
CTR-GCN	D1	16-64-0	0.157	0.681	0.643	0.631	0.861
CTR-GCN	D1	16-64-1	0.000	0.611	0.585	0.439	1.000
CTR-GCN	D1	16-64-2	0.001	0.595	0.450	0.427	1.000
CTR-GCN	D1	16-64-3	0.270	0.702	0.679	0.605	0.948
CTR-GCN	D1	64-64-0	0.174	0.693	0.665	0.650	0.861
CTR-GCN	D1	64-64-1	0.300	0.641	0.627	0.586	0.841
CTR-GCN	D1	64-64-2	0.280	0.625	0.650	0.580	0.833
CTR-GCN	D1	64-64-3	0.255	6.709	0.618	0.624	0.935
CTR-GCN	D2	16-64-0	0.000	0.635	0.526	0.465	1.000
CTR-GCN	D2	16-64-1	0.091	0.762	0.689	0.651	**0.960**
CTR-GCN	D2	16-64-2	0.000	0.697	0.605	0.535	1.000
CTR-GCN	D2	16-64-3	0.004	0.667	0.524	0.512	1.000
ST-GCN	D1	64-64-0	0.009	0.710	0.758	0.656	0.917
ST-GCN	D1	64-64-1	0.065	0.670	0.705	0.611	0.899
ST-GCN	D1	64-64-2	0.250	0.720	**0.773**	**0.713**	0.879
ST-GCN	D1	64-64-3	0.027	0.734	0.743	0.662	0.948

Table 7: Evaluation scores for models with the test set

3.3 Testing

Based on the validation scores, four models were picked for running the test set on. The models chosen are ones with the highest validation scores from each table in section 3.2. The test scores are presented in table 7. Their corresponding confusion matrices ROC-curves are presented in Appendix B. This is the first time the models are evaluated on the test set in order to keep with good machine learning practice.

4 Conclusions

From table 7 it is obvious that the ST-GCN model performs best with the given dataset. This is interesting since it is a predecessor to the CTR-GCN model and thus should be less developed. However, it is possible that this is the reason for the better performance. It could be that CTR-GCN picks out too many irrelevant features in a dataset as flawed as this one and that less features is a good thing because of this.

The dataset is very flawed. This is due to multiple factors, some discussed in section 2.1. Firstly, sloppily labelled data combined with a high fraction of poor quality data meant that the samples collected are varying in pertinence. If the labelled action's timestamp is off for more than a couple seconds the entire sample could end up mislabelled. Poor quality video footage makes it harder for the pose extractor to extract a complete and

correct skeleton and a lot of potentially useful samples had to be discarded because of the problem with unsynced videos. These issues could be solved by relabelling the data with higher precision and collect new data of higher quality. It would be time consuming and expensive but not impossible.

Secondly, finding a pose estimator that reliably put out high quality skeletons even on good data turned out near to impossible. Due to it not being the main issue of this project - the issue being big enough to warrant its own project - and almost all publicly available pose estimators having been tried earlier in the company's history, this project had to settle with the somewhat reliable and easily implemented MediaPipe pose estimator. Not being able to handle more than one person in a frame discarded a lot of potentially useful samples. Another big issue is that all footage is taken at an angle from the roof while all pose estimators found online are trained on levelled footage. This is impossible to solve save for training a custom pose estimator on angled footage which would be a huge and expensive undertaking. Obviously poor quality skeletons will effect performance immensely since they are the main input data.

Thirdly, due to lack of data from each store the models had to be trained in a very generic fashion. The angles of the cameras differ between all camera feeds, thus making it harder for the models to extract reliable features specific to one store or angle. The more angles provided the more general the model has to become, thus making good predictions harder. This problem is almost impossible to solve before a store has been open for long enough to train models on data provided from it. At that point, it might be too late to implement the model. A way of solving this could be to build a 3D-representation of the store in a game engine like Unity and create a dataset from modelled interactions. However, this would be very time-intensive and thus expensive, and it would have to be done for every store opened.

Furthermore, the datasets are too small to reliably train any deep neural networks. 1570 samples simply aren't enough for a model this complex. For comparison, one of the datasets used in the CTR-GCN paper has 114 thousand videos to train on (Code 2022). This issue could be solved with time or by the 3D-representation discussed in the paragraph above but the issues remains.

The flawed skeletons is the reason for trying compounded datasets (model version 0). The idea was that since all camera feeds show the same action but from different angles, the possibility of each frame providing any useful information increase if all three feeds are provided at the same time. The results from validation and testing doesn't prove or disprove this hypothesis in any deciding way but the best performing model in table 7 is not one trained on compounded data suggesting that the method doesn't provide any reliable benefits.

Finally, the outcome of this project was the realization that the CTR-GCN model might

not be the optimal model for Moonshop's Human Action Recognition. However, any conclusion like that is shadowed by the flaws in the dataset. A more relevant conclusion would be that trying to develop a reliable Human Action Recognition model based on skeletal data at this point would be a little like trying to run before one can walk. The models show potential so there could be a future for the technology but in order to get there, Moonshop has to take a couple of steps back and solve the data issues first. Once reliable skeletons can be extracted from a big, well-labelled dataset the methods developed in this project could be used to draw more substantiated conclusion.

All this focus on the flaws in the dataset could make it sound like the conclusion only explored excuses for a lack of results. However, that's not the case. A more thorough exploration of the dataset should have been done before any actual work took place, in order to determine whether the tasks were reasonable to tackle at all. As we've come to realize, such an exploration would probably have resulted in a pessimistic or negative conclusion. This would've allowed the focus to shift to a more realistic problem, possibly one of the problems suggested earlier in this section. This is one of the strongest lessons I will take with me from this project.

To tie things off, one relevant question is whether the tasks this project set out to complete are possible to solve at all, regardless of data quality and quantity. There might be a reason why other solutions on the market use more complex setups with scales and other measurements. Keeping track of multiple people, their actions in the store and interaction with possibly hundreds of wares is an enormous task. Limiting the data input to only a limited number of video feeds might be asking too much of AI at its present stage of development. It's not unreasonable to assume that such a system won't ever be 100 % correct. If that's the case, the question becomes one of business: at what level of precision is the gains greater than the losses this lack of precision cause? That question, however, is one best left to the capable hands of the Moonshop leadership.

References

Casalegno, Francesco (Jan. 2021). *Graph Convolutional Networks — Deep Learning on Graphs*. URL: https://towardsdatascience.com/graph-convolutional-networks-deep-99d7fee5706f.

Chen, Yuxin et al. (2021). "Channel-wise Topology Refinement Graph Convolution for Skeleton-Based Action Recognition". In: *IEEE*. DOI: 10.1109/ICCV48922.2021.01311.

Code, Papers With (June 2022). *NTU RGB+D 120 Dataset*. URL: https://paperswithcode.com/dataset/ntu-rgb-d-120.

Education, IBM Cloud (Oct. 2020). *Convolutional Neural Networks*. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

Al-Faris, Mahmoud et al. (2020). "A Review on Computer Vision-Based Methods for Human Action Recognition". In: *Journal of Imaging*. DOI: 10.3390/jimaging6060046.

Heller, Aron and Abhay Savargaonkar (Apr. 2021). *The rise in automation and what it means for the future*. URL: https://www.weforum.org/agenda/2021/04/the-rise-in-automation-and-what-it-means-for-the-future/.

Korstanje, Joos (Aug. 2021). *The F1 Score*. URL: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6.

MediaPipe (June 2022). *MediaPipe Pose*. URL: https://google.github.io/mediapipe/solutions/pose.

Narkhede, Sarang (June 2018). *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

PYMNTS (July 2021). *The 'Third Wave' Of Self-Serve Checkout Turns Grocery Stores Into Omnichannel Hubs*. URL: https://www.pymnts.com/news/retail/2021/the-third-wave-of-self-serve-checkout-turns-grocery-stores-into-omnichannel-hubs/.

Yan, Sijie, Yuanjun Xiong, and Dahua Lin (2018). "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition". In: *AAAI*. DOI: 10.48550/arXiv.1801.07455.

Zindi (2021). *Autonomous Shopper Prediction by Cape AI*. URL: https://zindi.africa/competitions/indabax-south-africa-2021.

Appendix A - Dictionary

- **Action**: A customer either taking a ware from a shelf or putting a ware back into a shelf. Presented in the basket dictionary.
- **Action Frame**: The specific frame labelled as the action in the basket dictionary. Calculated by a timestamp.
- **Basket**: A customer's visit to the store. Everything related to that visit is saved in the basket.
- **Basket-ID**: An identifier unique to every basket. Used, for example, to locate baskets in the different databases.
- **Batch**: A set of tensors. A DataLoader consists of equally sized batches.
- **Batch Size**: A variable determining how many tensors to keep in each batch.
- **Dataset**: Usually refers to the set of SkeletalKeypoint instances created by the data process pipeline.
- **Frame**: A picture of a movie, the smallest denominator. Presented as a matrix with pixel values.
- **Keypoints**: The joints of the human skeleton extracted by the pose estimator. In this project each skeleton has 33 keypoints.
- **Model**: A trained AI usable for whatever it was trained for. Sometimes also referred to the AI's architecture before it being trained.
- **Non-action Frame**: Any frame that's not an action frame.
- **Pose Estimator**: A model trained to extract human poses from images and represent them as skeletons through keypoints.
- **Raw Data**: The videos and dictionaries related to the baskets before they have gone through the data processing pipeline.
- **Store**: The physical location where the cameras were set up and the baskets were recorded.
- **Vendor**: The owners of the stores. Each vendor can own multiple stores each store can only have one vendor.

Appendix B - Plots

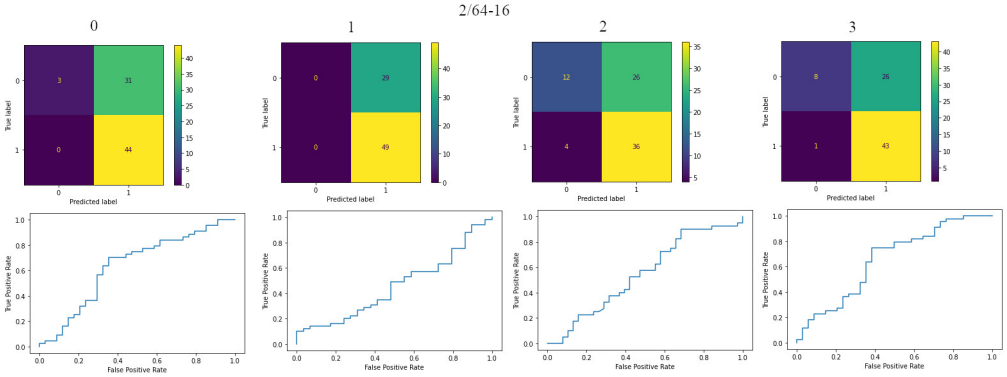

Figure 9: Plots with Validation Scores for CTR-GCN models with dataset D2

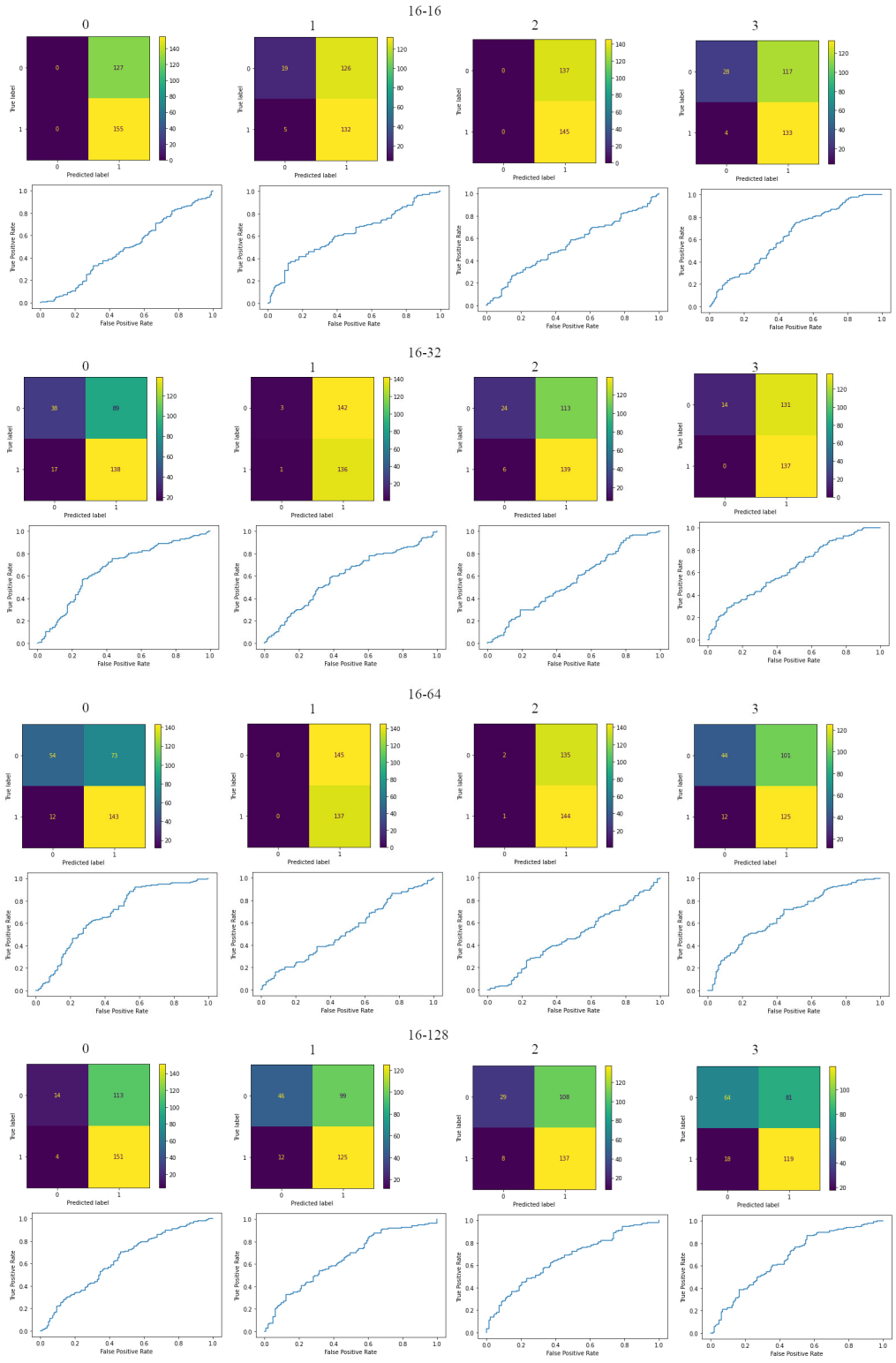

Figure 10: Plots with Validation Scores for CTR-GCN models with Batch Size = 16

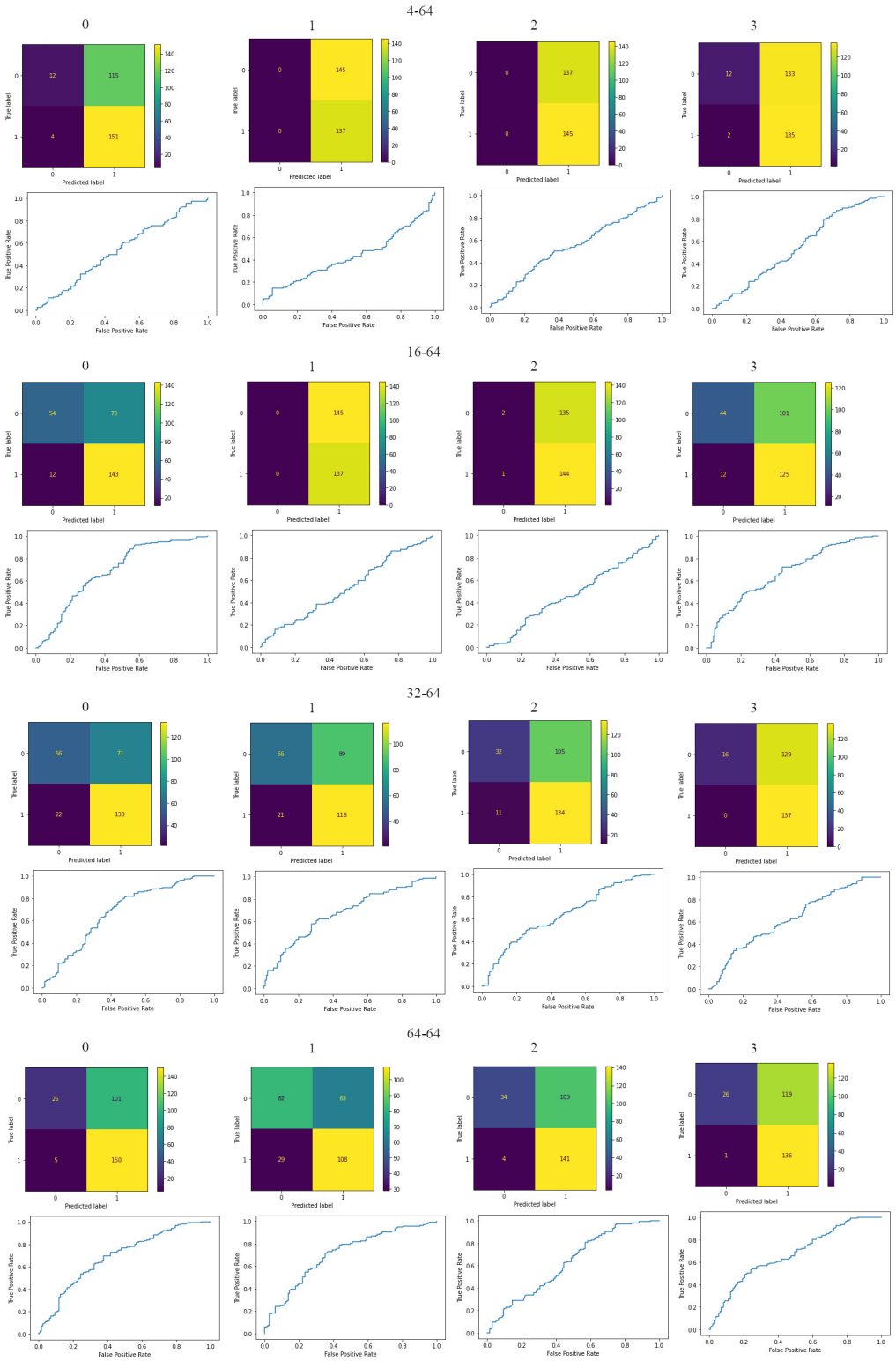

Figure 11: Plots with Validation Scores for CTR-GCN models with Base Channels = 64

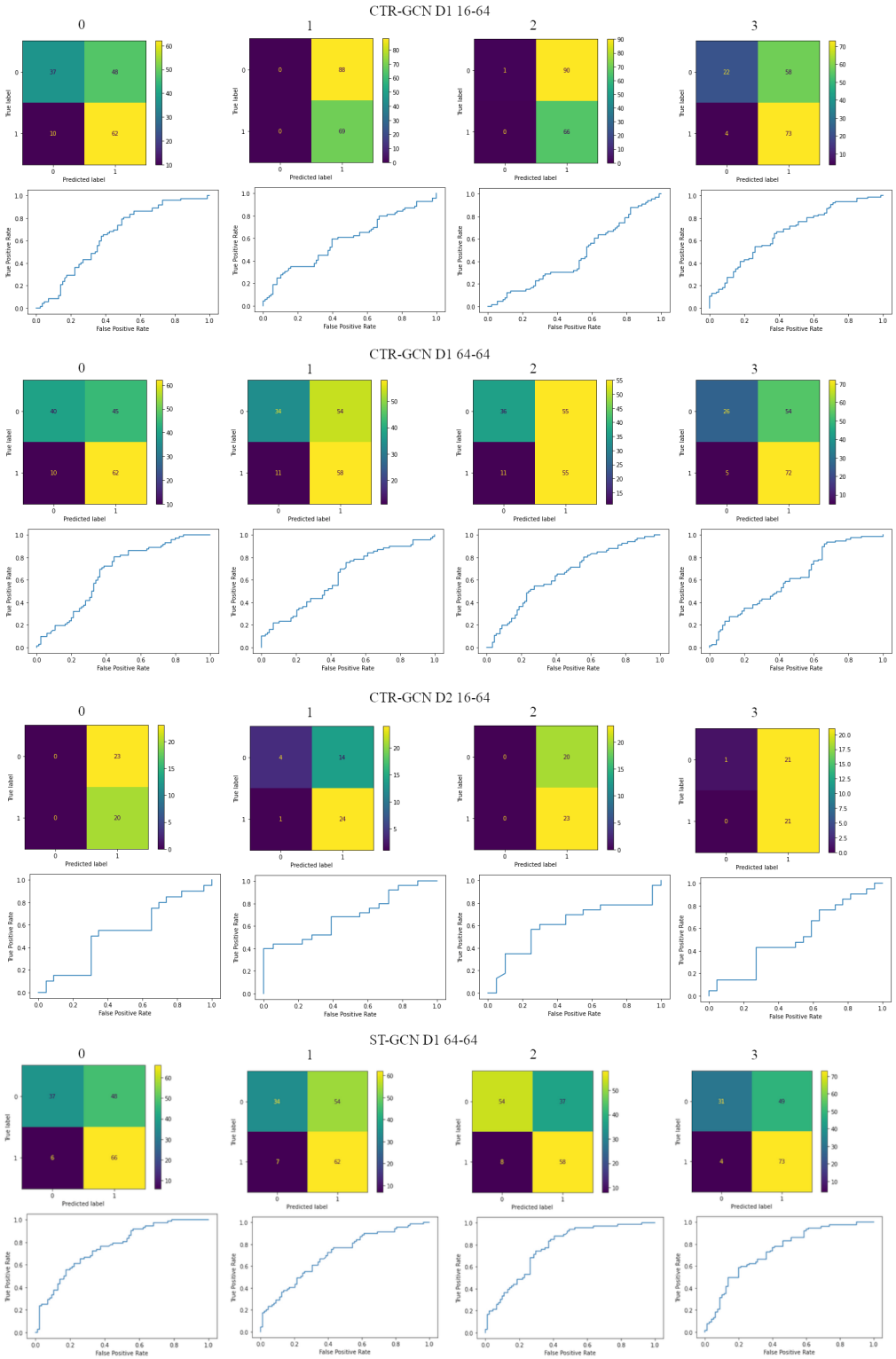

Figure 12: Plots with Test Scores