# Training and Deploying Computer Vision Models for Indoor Localisation

**Mika Senghaas**
IT University of Copenhagen
*jsen@itu.dk*

A Thesis presented for the Degree of
**Bachelor of Science in Data Science**

IT UNIVERSITY OF CPH

**IT University of Copenhagen**
Computer Science Department

May, 15th 2023

# Contents

**Abstract**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

# 1 Introduction

With the introduction of GPS (Global Positioning System), a satellite-based positioning system, localisation in outdoor spaces has become more efficient and accurate than ever before. Gradual commercialisation led to the technology rapidly transforming entire industries and personal navigation in every-day life. Today, outdoor localisation is widely considered a *solved problem.*

The same cannot be said for indoor localisation. Because the transmitted radio signals sent out by the satellites in GPS systems are not strong enough to penetrate through walls and struggle with reflections from large buildings, the technology yields inaccurate results at best, and often becomes dysfunctional in indoor spaces.

Finding alternative solutions to provide an accurate, cheap and robust indoor localisation systems has been a main focus of research in the past decades, and is becoming increasingly important in the light of the ongoing urbanisation of our living spaces and the emergence of autonomous robots and vehicles in our everyday life. Decades of research have led to the development of a variety of different indoor localisation technologies. Hardware-based systems use radio signals, transmitted by beacons, like Bluetooth, and Ultra-Wideband (UWB) or Wi-Fi, to localise an agent in a known environment. Software-based systems, like Simultaneous Localisation and Mapping (SLAM) algorithms, use sensors, like cameras or distance-measuring laser sensors, to localise an agent, while simultaneously creating a map of the environment.

While these approaches have proven to produce remarkable results, being capable of localising an agent with centimetre accuracy, they are limited for various reasons: Hardware-based system require an expensive initial setup, continuous maintenance of the signal-transmitting beacons, and are often not feasible in large environments. SLAM algorithms, on the other hand, require a meticulously handcrafted pipeline of feature detection, feature matching, and pose estimation that has to be fine-tuned by experts for each indoor space, to achieve outstanding results. All of the above short-comings of existing indoor localisation technologies make commercial applications rare and not unified in their approach.

All of the above approaches work under the assumption that centimetre-accuracy is required for indoor localisation. However, this is not always the case. For some applications, like indoor navigation, it is sufficient to know the position of the agent with a meter or even room-accuracy, instead of centimetre accuracy. One example of such an application is the navigation in large indoor spaces, like airports, train stations, or shopping malls, where the goal is to guide the agent to a specific room or area. In these cases, the constraint of centimetre-accuracy can be relaxed, in favour of a simpler and more versatile solution.

In the past decades, deep learning has proven to be a powerful tool in a wide-variety of tasks and has repeatedly proven remarkable capabilities in the field of computer vision. Amongst

common task in computer vision is the task of image classification where the goal is to predict a label from a set of pre-defined labels.

Motivated by the apparent lack of a simple, unified indoor localisation system and the success of deep learning in computer vision, this study investigates the applicability of modern deep learning techniques to the task of indoor localisation when viewing localisation as a classification task. The study presents the rigorous evaluation of several modern deep learning architectures on a challengingly small video dataset for mapping our a novel indoor space to imitate the real-world scenario of a new indoor space.

# 2 Background

Producing accurate, robust and cheap localisation systems is not a novel task, but has been a focus of research at the intersection of robotics, computer vision and machine learning for decades. A wide variety of approaches using different sensory information, assuming different agent and environment set-ups have been proposed.

## 2.1 Hardware-Based Indoor Localisation

The approach conceptually closest to GPS-based outdoor localisation systems are hardware-based indoor localisation systems. Because radio signals transmitted by satellites are incapable of penetrating through walls, various close-proximity radio signals have been proposed to be used for indoor localisation, such as Bluetooth [4, 6], Ultra-Wideband (UWB) [2, 3] or Wi-Fi [15, 17].

No matter the radio signals, three main approaches can be distinguished: (1) Angle of Arrival (AOA), (2) Time of Arrival (TOA) and (3) RSSI-Fingerprinting (RSSI-FP) [17].

In AOA approaches, transmitting beacons, called access points (APs), are measuring the distance and angle between a beacon and an agent. The intersection between the lines of sight (LoS) of at least three APs yields the position of the agent.

TOA approaches (such as GPS) use the received signal strength (RSS) to estimate the distance between an agent and the AP through a propagation model. Given the distance estimates of at least four APs, the agent's position can be determined by an approach called tlateration, which is based on the idea that an agent's position in three-dimensional space can be uniquely determined by the intersection of four spheres with known radius and centres. Clearly, TOA approaches require knowledge about the position of the APs, which makes them less practical in indoor spaces, where the position of the APs is often unknown.

The approach with the most recent research interest is RSSI-FP. It can be divided into two separate phases: In an offline mapping phase, the indoor environment is mapped by repeatedly measuring the RSSI values of the APs at various reference points (RPs). A vector of RSSI values then uniquely identifies each RP and is stored in a database. In the online localisation phase, the agent's position is then determined by dynamically comparing the agent's RSSI vector against the RSSI vectors of the RPs stored in the database. Ultimately, the agent's position is determined as the position of the RP with the most similar RSSI vector according to some similarity metric. While this approach alleviates the need for the precise positions of the APs, it requires a time-intensive offline mapping phase and is sensitive to changes in the environment.

Many of the proposed technologies assume massive infrastructure deployments and incur high setup and maintenance costs. Wi-Fi based approaches seem most promising, given their ubiquitous availability in modern indoor spaces. However, all hardware-based approaches suffer measurements errors that underlie the triangulation (AOA), multilateration (TOA) or similarity metric (RSSI-FP) approaches. Such errors are caused by reflections (multipath effect), blockage (shadowing), and signal attenuation (fading).

## 2.2 Simultaneous Location and Mapping (SLAM)

Amongst the most promising approaches are SLAM (Simultaneous Localisation and Mapping) algorithms. SLAM algorithms aim to localise an agent inside an unknown environment, while simultaneously building a consistent map of the environment. There exist a variety of different approaches to SLAM, depending on the type of sensors that are used. For example, Visual SLAM (V-SLAM) algorithms use camera input, and LidarSLAM algorithms use distance-measuring laser sensors.

Most related to this study are monocular V-SLAM algorithms, because they use a single camera to estimate the position of the agent. The very first monocular feature-based V-SLAM algorithms, MonoSLAM, was proposed in 2007 [9]. The research is considered a break-through in V-SLAM algorithms, as it is considered the first algorithm producing accurate results while only using a single camera. Previously, V-SLAM algorithms required multiple cameras or other sensors to overcome the problem of depth estimation using a single camera.

Since then, many adjustments and optimisation have been proposed to the algorithm to make it more robust and accurate. Typically, the adjustments replace or modify one of the components of the pipeline. For example, the ORB-SLAM [25] algorithm uses a bag-of-words approach for feature matching, and the PTAM [18] algorithm uses a parallel computing, which was shown to improve the accuracy of the algorithm. In recent years, such modifications are often based on deep learning techniques to improve on sub-parts of the SLAM pipeline. For example, the DeepVO [31] algorithm uses a convolutional neural network to estimate the camera pose from a sequence of images.

Overall, SLAM algorithms have shown impressive results and are in commercial use in many applications. However, they are still not robust enough to be deployed in safety-critical applications, such as autonomous driving. The

## 2.3 Deep Learning in Computer Vision

Computer vision is one of the major subfields of artificial intelligence and is concerned with the automatic extraction of information from images. The extraction of information from images is a challenging task, because images are high-dimensional and unstructured. This makes it a challenging environment for hand-crafted algorithms, which are often based on heuristics and assumptions about the data. For this reason, deep learning techniques have been applied successfully to many computer vision tasks, and dominate the benchmarks of most computer vision tasks.

Arguably the offset of the deep learning wave in computer vision was the success of the AlexNet [21] architecture in the 2012 ImageNet Large Scale Visual Recognition Challenge [10], which crushed the competition by increasing the Top-5 Accuracy from 73.8% to 84.7%. At the time, the dominating approach to computer vision tasks was based on hand-crafted features, such as SIFT [24] and HOG [8]. AlexNet was the first successful adaption of a deep convolutional neural network (CNN), which were first proposed by LeCun et al. [22] in 1998.

In the following years, CNN-based architectures with increasing depth and width, and architectural improvements, such as weight initialisation, batch normalisation, residual connections [27, 28, 14, 7] were proposed and pushed the state-of-the-art further. The introduction of ResNet in 2015 is considered a cornerstone in convolutional neural networks architectures, as it allowed for the training of much deeper networks without the problem of vanishing gradients. In 2017, the Xception [7] architecture was proposed, which is based on the Inception architecture, but uses depth-wise separable convolutions.

With the success of Transformer-based architectures for language modelling in natural language processing [30], the computer vision community was quick to adapt the architecture to

computer vision tasks. The first successful application of the Transformer architecture to computer vision was the Vision Transformer (ViT) [12], which achieved state-of-the-art on many image classification benchmarks [19, 20, 10].

Beyond image classification, the task of video classification has recently gained traction with the introduction of large-scale video datasets, such as Kinetics [16], more computational resources and an increasing need to understand video data in our multi-media world. Continuously predicting on a stream of frames is clearly related to image classification, but adds complexity to the task, because of the temporal dimension and increase in data size. Video Classification can be solved by combining CNNs with recurrent neural networks (RNNs) or by using 3D convolutions. The first approach by Donahue et. al proposes long-term recurrent convolutional networks (LRCNs), which use a CNN to extract features from each frame, which are then fed into a LSTM [11] to produce a sequence of predictions. Alternative approaches use 3D convolutions to capture the spatio-temporal nature of video data [29, 5]. Yet another approach, called SlowFast [13] uses two separate CNN networks to capture spatial and temporal information separately, which are then combined in the final layers of the network.

# 3 Data

Framing the problem of indoor localisation as a classification task requires a labelled data set, which consists of sequentially-arranged pairs of video frames and location labels. A single video clip $C$ consists of a sequence of $n$ frames $x_0, ..., x_n$ and a sequence of $n$ location labels $y_0, ..., y_n$, where the $i$-th pair of the sequence is the tuple $(x_i, y_i)$ and denotes the location at a specific frame. A single frame $x_i$ is a RGB image, represented as a three-dimensional tensor of shape $3 \times H \times W$, where $H$ and $W$ are the height and width of the image, respectively. A single location label $y_i$ is a scalar value, which identifies the location of the agent at the time of the frame.

## 3.1 Data Collection

The data set was collected from a single camera of a mobile device [1] that was hand-held by a human agent while walking around an indoor building. The chosen location for the data collection was the main building of the Southern Campus of the Copenhagen University (Danish: Københavns Universitet, KU) in Copenhagen S, Denmark. The building is a large multi-storey building with a total of six floors, and is used for teaching and research purposes. The location was deemed compatible with this study, as it both has distinctive indoor features (e.g. coloured walls, unique structures, etc.), but also poses a challenge for the model, for example, due to the similarity of the floor plan across floors. For the scope of this project, the data collection was limited to the first two floors. The publicly accessible areas were separated into 21 different location labels, which are shown in Figure 1. Whenever possible, the location labels were designed in close correspondence to the building's official floor plan, but given more descriptive names.

A total of 53 video clips were recorded, with an average duration of $\sim$ 57s, amounting to a total number of $\sim$ 50 minutes of footage. Out of the total 53 video clips that were recorded, 37 were used for training and 16 were used for validation. Table 1 shows more statistics of the raw data in the two different splits.

An ideal model learns robust indoor features that are invariant to natural variations in the indoor space from training data collected in a minimal time span, as this minimises the initial cost and effort for data collection. To evaluate the model's robustness towards these variations, a different philosophy was adopted for the collection of training and testing data.

[1] iPhone 11 Pro recording at 30 FPS with 2426x1125 HD resolution

| **Split** | Total Clips | Total Seconds | Total Minutes |
|---|---|---|---|
| Training | 37 | 2240 | 37 |
| Testing | 16 | 783 | 13 |

Table 1: Statistics of the Raw Data in Training and Testing Splits

While the 37 training clips were recorded on just two days, the 16 testing clips were recorded on four different days, two to four weeks after the initial training data collection. This was done to make the testing data closer resemble real-world scenarios, and thereby make the testing metrics more robust.

### 3.2 Data Annotation

To match the location labels to the video footage, each clip $C$ was manually annotated by denoting the starting and ending time stamps of a location label. The information was stored in a standardised format and later used in the pre-processing of clips into frame-location pairs.

### 3.3 Data Preprocessing

The raw video clips and manual annotation of location labels, had to be pre-processed before they could be used for training deep learning models.

The video footage was resized to a resolution of 224x224 pixels, which is the input resolution of most modern foundation models for image and video classification, which were used in this project. Furthermore, it decreases the total data amount significantly, which allowed for less disk usage and faster loads into memory during training.

Next, instead of extracting all 30 frames per second, the video footage was downsampled to a much lower frame rate of 1 FPS. This was done to reduce the total data amount, and to reduce over-fitting of the model to the training data. It was hypothesised, that because of the strong local dependency of consecutive frames, consecutive frames are highly correlated, thus including such frames does not introduce any additional variance, but redundancies that are harmful for the model's generalisation ability. Empirical experiments confirmed that downsampling indeed reduces over-fitting, and was therefore adopted across all experiments.

Finally, the pre-processed video footage was aligned with the manual location labels by extracting a frame per second from the video and matching it against the location label that was active at that time. After pre-preprocessing, there exist pairs $(x_i, y_i)$, where $x_i$ is pre-processed, extracted frame of size 224x224 pixels, and $y_i$ is the corresponding location identifier. The frames were stored in a way that allowed for easily sampling both a random frame-location pair $(x_i, y_i)$ for image classification models, or a sequence of consecutive $n$ frames from a clip $([x_0, \ldots, x_n], [y_i, \ldots y_n])$ for video classification models. Here the sequence of frames was represented as a 4D-tensor of size $n \times 3 \times 224 \times 224$.

Figure 2 shows $n = 4$ consecutive frame-location label pairs after preprocessing. As can be seen, even at a frame rate of 1 FPS, neighbouring frames are still similar to each other. Further, the annotation at the transition from one location to another was often difficult to determine, as one could annotate strictly according to the device position or the view of the camera. This likely resulted in some frames being annotated with the wrong location label.

# 4 Methodology

## 4.1 Models

A deep learning model is supposed to learn a function $f : X \to Y$, where $X$ represents the input space and $Y$ the output space. By having framed the task of indoor localisation as a classification problem, the output space is the discrete set of location labels, $Y = \{l_1, \ldots, l_n\}$. Two approaches are viable for the input space $X$:

1. **Image Classification**: The input space $X$ is a single frame, $(3, H, W)$ and the output space is a single location label $y \in Y$. The model treats consecutive frames independently, and therefore disregards the temporal dimension of the video.

2. **Video Classification**: The input space $X$ is a sequence of $N$ frames, $(x_1, \ldots, x_n)$, and the output space $Y$ is a sequence of $N$ location labels, $(y_0, \ldots, y_n)$, where each $y_i \in Y$. The model considers the temporal dimension of the video, and learns to classify the location label of a sequence of frames. In reality, the input sequence has a maximum context length $K$, meaning that the model continuously predicts on the previous $K$ frames.

Figure 3 shows the high-level architecture of the two approaches. For an image classification model, a sequence of $N$ (here $N = 4$) frames are independently fed into a convolutional neural network (CNN), which outputs a feature vector for each frame. CNNs are a class of deep neural networks that are applied to analyse visual imagery. They are composed of several layers of alternating convolutional and pooling filters, which are applied to the input image to extract high-level feature representations. These high-level feature representations are then fed into a fully connected neural network (FCNN), which outputs a probability distribution over the location labels.

For a video classification model, all $N$ consecutive frames are independently passed through a CNN module. The sequence of high-level feature representations is then fed into a recurrent neural network (RNN), which outputs a probability distribution over the location labels for each frame by placing a fully connected layer (classification head) to the output vector associated to each frame in the RNN module. A recurrent neural network (RNN) is a class of deep neural networks that is applied to analyse sequential data. Although they are mostly applied to natural language processing tasks, the generality of their architecture allows to also apply them to computer vision tasks with sequential data, such as a sequence of frames in video classification. For each input $x_i$ in a sequence of inputs, the RNN computes an output as a function of the current input $x_i$ and a hidden state $h_{i-1}$ that is computed as a function of all previous inputs. In that way, the RNN can learn to model the temporal dependencies between frames, e.g. consider that all previous frames were predicted to be at one specific location, and therefore the current frame is also likely to be at that location.

To evaluate the performance of both approaches, a selection of well-performing CNN and RNN modules were chosen, and combined to form different image and video classifiers. The modules, alongside meta information about their number of parameters, size in Megabyte (MB) and GFLOPS (Giga Floating Point Operations per Second) are provided in Table 2. Generally speaking, smaller model sizes are preferred, as they are faster to train and during inference and require less memory, making them more suitable to be deployed on low-resource devices like mobile phones and embedded devices. It is expected that more complex, larger models will perform better, so the goal is to choose the smallest model that works well enough to perform the task with sufficient accuracy.

By replacing the final fully connected layer of the CNN with a linear layer with $N$ outputs, where $N$ is the number of location labels, each CNN module can be used as an image classifier.

| Type | Name | #Params | GFLOPS | Size (MB) |
|---|---|---|---|---|
| **CNN Module** | Resnet18 [14] | 11.6M | 1.81 | 44.7 |
| | Resnet50 [14] | 25.6M | 4.09 | 99.8 |
| | MobileNet-V3 | 2.5M | 0.06 | 9.8 |
| | Alexnet | 61.1M | 0.71 | 233.1 |
| **RNN Module** | RNN | 0.26 | N/A | 1.04 |
| | LSTM | 1.05 | N/A | 4.2 |

Table 2: CNN and RNN Modules

Because the different RNN modules expect a one-dimensional input for each element in the input sequence of frames, the RNN modules are combined with a CNN module to form a CNN-RNN architecture. Given the 4 different CNN modules and the 2 different RNN modules, 8 different CNN-RNN architectures were formed for the video classification approach. The list of all image and video classifiers that were evaluated is provided in Table 3.

| **Name** | **CNN Module** | **RNN Module** |
|---|---|---|
| resnet18 | ResNet18 | - |
| resnet50 | ResNet50 | - |
| mobilenet_v3_small | MobileNet-V3 | - |
| alexnet | AlexNet | - |
| resnet18-rnn | Resnet18 | RNN |
| resnet50-rnn | ResNet50 | RNN |
| mobilenet_v3_small-rnn | MobileNet-V3 | RNN |
| alexnet-rnn | AlexNet | RNN |
| resnet18-lstm | Resnet18 | LSTM |
| resnet50-lstm | ResNet50 | LSTM |
| mobilenet_v3_small-lstm | MobileNet-V3 | LSTM |
| alexnet-lstm | AlexNet | LSTM |

Table 3: List of all Models

## 4.2 Training

All models were implemented using the PyTorch framework [1] and the training was performed locally on a MacBook Pro M1 with 16GB of memory and MPS-acceleration enabled (where possible[2]) to allow for fast experiment iteration. The loss function $\mathcal{L}$ used for all models was cross-entropy loss (Equation 1), which is a standard loss function for multi-class classification problems.

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{1}$$

Here, $\hat{y}$ is the predicted probability distribution over the $K$ classes and $y$ is the one-hot encoded ground truth label. The models were trained using the AdamW [23] optimiser with

[2]MobileNet-V3-based models do not support GPU acceleration

default parameters, except for the learning rate, which was set to a constant of $1e^{-4}$. Step-wise learning rate scheduling was used for all models, which reduced the learning rate by a factor of 10 every 5 epochs. The batch size was set to 32 for all image classifiers and 8 for all video classifiers for memory-optimal training. Unless otherwise specified, all image and video classifiers were trained with the same set of training hyper-parameters, which are specified in Table 4.

| Classifier | Batch Size | Epochs | Optimiser | Learning Rate Scheduler |
|---|---|---|---|---|
| **Image** | 32 | 10 | AdamW ($\gamma = 1e^{-4}$) | Step-LR ($\gamma = 1e^{-1}, s = 5$) |
| **Video** | 8 | 10 | AdamW ($\gamma = 1e^{-4}$) | Step-LR ($\gamma = 1e^{-1}, s = 5$) |

Table 4: Default Hyperparameters for Image and Video Classifiers

### 4.3 Evaluation

The goal of the evaluation is to have a rigorous comparison of the different models and to determine the best model for the task of location classification. In this context, a "well-performing" model is not just the model that is most accurate in most cases, but also the model that is most robust to a wide-range of different natural variations that can occur and that is efficient. To achieve this, a series of *quantitative* and *qualitative* experiments are performed on the models, which are then used in Section 5 and Section 6.

**Quantitative Experiments.** To quantitatively assess the performance of the model on the task of location classification, the models were evaluated on the test set using a wide-range of different performance metrics. To assess the overall accuracy of the model, two metrics are computed: **Multi-class Accuracy** (Equation 2) is the most wide-spread performance metric in classification task and is defined as the number of correct predictions divided by the total number of predictions. It gives a good first indication for the overall performance of the model, but can be misleading in cases where the dataset is imbalanced.

$$\text{Multi-class Accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(y_i = \hat{y}_i) \tag{2}$$

Here, $\mathbb{I}$ is an indicator that is 1 if the condition is true and 0 otherwise. On top of the standard, multi-class accuracy, a variant of the formula, the **Top-3 Multi-class Accuracy** is also computed, which counts a prediction as correct if the correct label is one of the top-3 predicted labels.

As some location labels are underrepresented in the dataset due to the natural variation in size of the different locations, the **Macro F1-Score** is used to compute a more fine-grained metric. The Macro F1-Score is the average of the class-specific F1-scores, which are defined as the harmonic mean of precision $P_i$ and recall $R_i$ (Equation 3).

$$\text{Macro F1-Score} = \frac{1}{N}\sum_{i=1}^{N}\frac{2 \cdot P_i \cdot R_i}{P_i + R_i} \tag{3}$$

Here, $P_i$ and $R_i$ are the precision and recall for class $i$, which are defined as follows (Equation 4) and (Equation 5).

$$P_i = \frac{TP_i}{TP_i + FP_i} \tag{4}$$

$$R_i = \frac{TP_i}{TP_i + FN_i} \tag{5}$$

Here, $TP_i$ is the number of true positives for class $i$, $FP_i$ is the number of false positives for class $i$ and $FN_i$ is the number of false negatives for class $i$.

On top of the standard metrics for classification, efficiency of the model is crucial for usability of the system on low-resource devices, such as mobile phones. For this reason, the **Inference Time** per sample is measured, the **GFLOPS** per predicted sample is computed and the **Model Size** is measured to assess the memory constraints imposed by the model.

**Qualitative Experiments.** Purely quantitatively assessing a model's performance may not be sufficient to truly determine the strengths and weaknesses of the model. For this reason, the models were also evaluated qualitatively by manually inspecting the misclassified samples and the predictions of the models on a sub-set of 20 test frames. Furthermore, the `GradCam` [26] algorithm was used to gain insights into the internal functioning of the model. GradCam is a technique that backtracks the activations in the convolutional filters of the model at some depth in the model's architecture and through the gradients of the model to the input image. This allows to visualise the regions of the image that were most relevant for the model to make its prediction. In this scenario, it is hoped that the highlighted regions can be used to understand what types of features the model is looking for in the image and what types of features it is not looking for. This can be used to identify potential issues with the model and to gain insights into how the model can be improved.

# 5 Results

The series of experiments conducted within this study show that computer vision models, when carefully designed and trained, are generally capable of solving the task of indoor localisation when phrased as a coarse-grained classification problem. Table 5 shows the results of the performance and efficiency metrics of all models (see Table 3).

| | **Model** | **1-Acc.** (%) | **3-Acc.** (%) | **Ma.-F1** (%) | **Par.** (M) | **FLOPs** (G) | **Throughput** (Preds/s) |
|---|---|---|---|---|---|---|---|
| Image | `alexnet` | 60.15 | 84.42 | 55.95 | 57.09 | 0.71 | **87.31 (± 1.15)** |
| | `resnet18` | 71.39 | 91.70 | **68.29** | 11.19 | 1.82 | 64.12 (± 1.90) |
| | `resnet50` | 58.37 | 82.63 | 55.92 | 23.55 | 4.12 | 27.57 (± 1.10) |
| | `mobilenet_v3` | 29.37 | 64.11 | 14.87 | 1.54 | 0.06 | 16.49 (± 1.44) |
| Video | `alexnet-lstm` | 32.60 | 62.19 | 9.95 | **58.58** | 3.57 | 38.31 (± 0.53) |
| | `alexnet-rnn` | 50.00 | 78.90 | 35.71 | 58.19 | 3.56 | 39.09 (± 0.51) |
| | `resnet18-lstm` | **72.19** | 91.78 | 64.59 | 11.84 | **9.11** | 19.66 (± 0.40) |
| | `resnet18-rnn` | 70.14 | **93.97** | 66.62 | 11.44 | 9.11 | 19.07 (± 0.56) |

Table 5: **Results.** The table shows a series of performance and efficiency metrics for the models trained in the first experiment. Here, `1-Acc` and `3-Acc` refer to the Top-1 and Top-3 Accuracy scores respectively, `Ma.-F1` refers to the Macro-F1 score, `Par.` refers to the number of parameters in the model, `FLOPs` refers to the number of floating point operations required to make a prediction and `Throughput` refers to the number of predictions that can be made per second. The values for `Throughput` are averaged over 10 runs and the standard deviation is reported in parentheses. The best performing model for each metric is highlighted in bold.

## 5.1 Detailed Analysis

The best performing model in terms of classification performance are the Resnet18-based models. The pure CNN module reaches a maximal Macro F1-Score of 68.29% and a Top-1 Accuracy of 71.39%. Stacking a RNN leads to slight decreases in the performance (-1.67%, -1,25%), while stacking a LSTM layer leads to slight increases in Top-1 Accuracy (+0.8%) and a more significant decrease in Macro-F1 (-3,7%). ResNet50 performs worse than ResNet18, with a maximal Macro-F1 of 55.92% and a Top-1 Accuracy of 58.37%. The increased number of parameters does not seem to be necessary given the complexity (and small size) of the dataset. Despite the higher number of parameters, AlexNet performs worse than ResNet18, with a maximal Macro-F1 of 55.95% and a Top-1 Accuracy of 60.15%. The architectural differences, specifically the lack of residual connections, are hypothesised to be the major cause for the drop in performance. Interestingly, by stacking a RNN or LSTM layer on top of the AlexNet module, classification performance drops sharply. With a RNN, the Macro F1-Score drops to 35,71% (-20.24%) and with a LSTM, the Macro F1-Score drops to 9.95% (-46.00%), barely better than a random guess. MobileNet-V3 performs the worst, with a maximal Macro-F1 of 14.87% and a Top-1 Accuracy of 29.37%. Given the significantly lower number of parameters (1.54M), the model does not seem complex enough to learn meaningful features in the dataset.

## 5.2 Performance Efficiency Trade-Off

The general trend in deep learning is that increasing the number of parameters, and thereby the complexity of the model, generally leads to better performance. In this experiment, higher model complexity was reached in two ways: Firstly, by increasing the number of parameters in the CNN module, and, secondly, by stacking a RNN or LSTM layer on top of the CNN module. In order to investigate the performance-efficiency trade-off, the number of parameters, the number of floating point operations and the throughput of all models were measured and plotted against the Top-1 Test Accuracy. The results are shown in Figure 5.

Figure 5 shows the Top-1 Test Accuracy of all models against the number of parameters.

It can be seen that higher model complexity generally does not lead to higher performance. Both ResNet50 and AlexNet have significantly more parameters than ResNet18, but perform worse. The picture is slightly different for the number of floating points operations and throughput. As expected, ResNet50, due to its higher number of parameters, requires more floating point operations during inference, which leads to a lower throughput. However, AlexNet, despite being almost 6x larger than ResNet18 in terms of pure parameter count, has a lower number of floating point operations and a higher throughput. However, since the base ResNet18 model already has a throughput of 65 frames per second, which is more frames per second than most modern cameras capture, the throughput does not seem to be a limiting factor.

## 5.3 Temporal Dimension

One of the main difference in the trained models is whether or not they are capable of processing the temporal nature of the video frames. It was hypothesised that the CNN-RNN video classification models would perform better due to an increased number of features and the ability to learn temporal information from the video frames. The results show that this is not the case.

Figure 5 not only shows the performance-efficiency trade-off, but also the difference in performance and efficiency between the base CNN modules and the CNN-RNN models. The grey lines connect the base CNN module with the CNN-RNN model. It can be seen that adding the RNN head leads to unchanged (ResNet18) or decreased (AlexNet) performance, and significantly higher number of floating point operations and therefore lower throughput. The analysis clearly

shows that adding a RNN head does not yield the desired effect of modelling the temporal dimension of the video frames to increase the robustness of continuous predictions. Instead, it merely increases the model complexity and inference time, which make it overall inferior to their pure CNN counterparts.

### 5.4 Data Efficiency

Approaching indoor localisation as a classification task, means to frame it as a supervised learning problem. Supervised learning requires human-annotated training data, which can be prohibitive to obtain in large quantities. For the proposed approach to be valid, minimal effort in the initial data collection and annotation should be required. For this reason, the second experiment tries to find an answer for the research question: *"How much data is necessary to achieve good performance"*. To answer this question, a single image and video classifier are chosen and trained on different subsets of the training data. The subsets include 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the training data. The models are then evaluated as outlined in Section 4.3 to find the natural drop-off in performance as the amount of training data decreases.

### 5.5 Analysing Failing Scenarios: Mispredicted Samples

To better understand the strengths and weaknesses of the best-performing model, a pure ResNet18 CNN module, the confusion matrix is analysed. Figure 7 shows the confusion matrix of the ResNet18 model.

### 5.6 Extracting Characteristics of the Model using GradCam

### 5.7 Deployment on Mobile Devices

As a proof of concept, the best-performing model, was deployed on a mobile device. The trained model was first quantised to 8-bit float precision and then converted to `TorchScript` format. The `TorchScript` model was then deployed on a mobile device using the `PlayTorch` framework *[CITE]*. The `PlayTorch` framework is a port of the PyTorch Mobile SDK that is natively compatible with native iOS and Android Deployment to Javascript. This allows for rapid prototyping and deployment of PyTorch models on mobile device using the popular React Native framework and Expo. The model runs locally on the device and therefore does not require any internet connection after initially downloading the model upon opening the app.

## 6 Limitations & Future Work

This study has shown the potential of using a pure deep learning pipeline for tackling the problem of indoor localisation. However, there are still many open questions that need to be addressed in future work for systems similar to the approach suggested here to be widely useful in real-world applications.

The main drawback of the proposed approach is the coarse location labels. State-of-the-art indoor localisation systems are able to localise users to centimetre accuracy. Therefore, future work should focus making improvements on the entire pipeline that allow for higher precision in the location labels.

Furthermore, this study assumes a small data set of only 40 minutes of video footage for training. While the experiment setup allowed to draw conclusion about the data efficiency of the models and the general feasibility of the data collection and annotation process, it is an

open question how similar systems scale to even larger indoor spaces with more training data available.

Additionally, it has to be noted that the test split was collected over a duration of four different days in close succession. Therefore, the test split might not be representative of the true variation of visual inputs from a indoor location over the course of a year. For example, the location might change more significantly than represented in the test split over seasons. In that case, the reported test metrics are likely to be over-optimistic. To gain confidence in favour or against this hypothesis, monitoring the performance of the trained models on test data collected over a longer period of time would be necessary.

Finally, the detailed analysis of the mispredicted samples has shown that most errors, because the models that show a lot of similar features, such as different libraries, or rooms that are architecturally similar across floors. Future work should specifically improve on finding solutions to these issues. Possible starting points might be to use the fact that sudden jumps in the building are impossible, so highly confident predictions from the past can be used as a strong indicator for the next room if some knowledge about the relative position of the rooms is available.

# 7 Conclusion

We have shown that the problem of indoor localisation can be phrased as a classification problem and be solved through the use of deep learning models in a data-efficient manner. Simple convolutional neural architectures were able to provide accurate and robust predictions and showed emerging learning of features mostly invariant to natural variations in the environment. The use of a recurrent neural network architecture allowed for the incorporation of temporal information and improved the accuracy of the predictions, though only marginally so.

# References


[1] Pytorch. https://pytorch.org.

[2] Nayef Alsindi, Bardia Alavi, and Kaveh Pahlavan. Measurement and modeling of ultrawideband toa-based ranging in indoor multipath environments. *Vehicular Technology, IEEE Transactions on*, 58:1046 – 1058, 04 2009.

[3] Nayef Alsindi and Kaveh Pahlavan. Cooperative localization bounds for indoor ultra-wideband wireless sensor networks. *EURASIP Journal on Advances in Signal Processing*, 2008, 04 2008.

[4] U. Bandara, M. Hasegawa, M. Inoue, H. Morikawa, and T. Aoyama. Design and implementation of a bluetooth signal strength based location sensing system. In *Proceedings. 2004 IEEE Radio and Wireless Conference (IEEE Cat. No.04TH8746)*, pages 319–322, 2004.

[5] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

[6] Sudarshan S. Chawathe. Beacon placement for indoor localization using bluetooth. In *2008 11th International IEEE Conference on Intelligent Transportation Systems*, pages 980–985, 2008.

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[9] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[13] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. *CoRR*, abs/1812.03982, 2018.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] Suining He and S.-H. Gary Chan. Wi-fi fingerprint-based indoor positioning: Recent advances and comparisons. *IEEE Communications Surveys and Tutorials*, 18(1):466–490, 2016.

[16] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

[17] Ali Khalajmehrabadi, Nikolaos Gatsis, and David Akopian. Modern WLAN fingerprinting indoor positioning methods and deployment challenges. *CoRR*, abs/1610.05424, 2016.

[18] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[20] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[23] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[24] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[25] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *CoRR*, abs/1502.00956, 2015.

[26] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[29] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[31] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. *CoRR*, abs/1709.08429, 2017.

# 8 Appendix

## 8.1 Reproducibility

All code and data used in this project is available on GitHub at the following link: `https://github.com/mikasenghaas/bsc`. The project's `README` file contains detailed instructions on how to reproduce the results of this project.

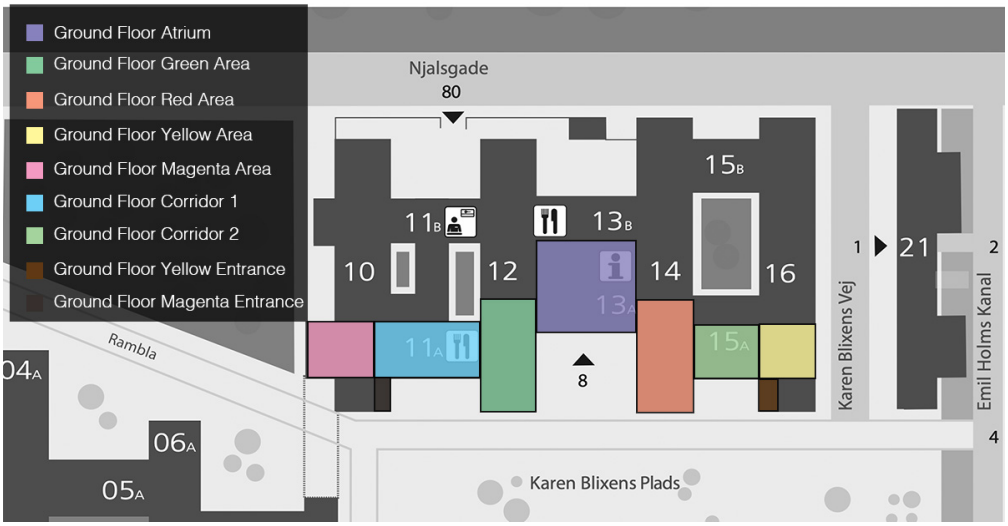
Further, the precise configuration and results of the experiments that are reported here are publicly available on the Weights & Biases platform at the following link: `https://wandb.ai/mikasenghaas/bsc`.

## 8.2 Machine Specifications

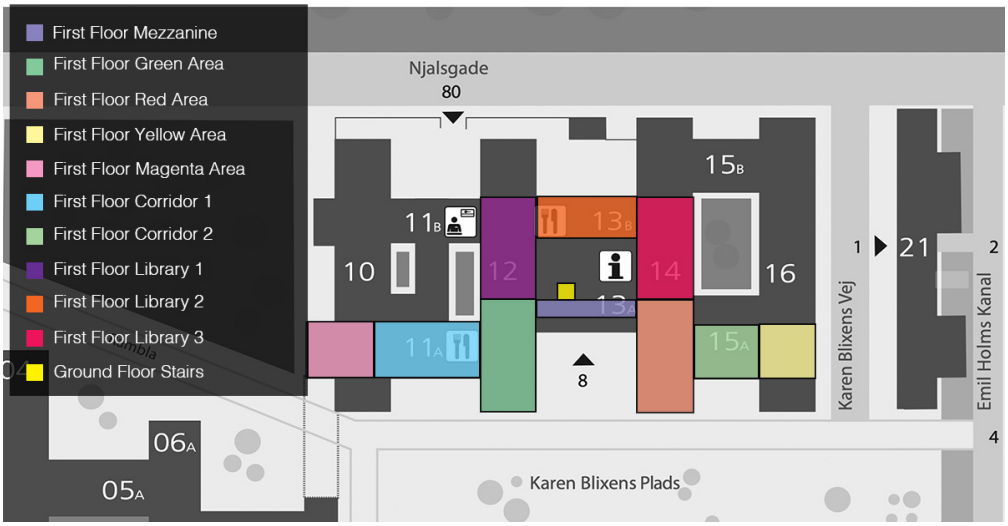
Table 6 lists the specifications of the machine that was used for training and evaluation of the models. For training the `MPS` backend was used for accelerated training. However, for inference a single CPU core was used, to approximate latency and throughput on mobile devices.

| | Specification | Value |
|---|---|---|
| System | Name | Darwin |
| | Node | Local MacBook Pro |
| CPU | Model | Apple M1 |
| | Architecture | ARM64 |
| | Physical Cores | 8 |
| | Frequency | 2.4 GHz |
| Memory | Total Capacity | 16 GB |
| | Avg. Used Capacity | $\sim$ 7.4 GB |

Table 6: Machine Specifications




(a) Ground Floor




(b) First Floor

Figure 1: Map of KU Southern Campus Main Building with Location Labels

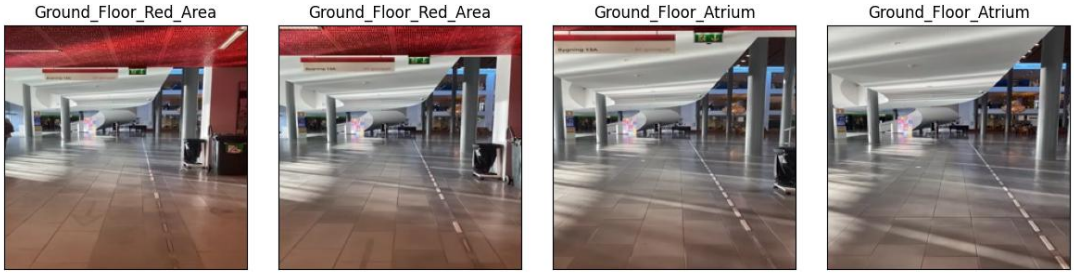
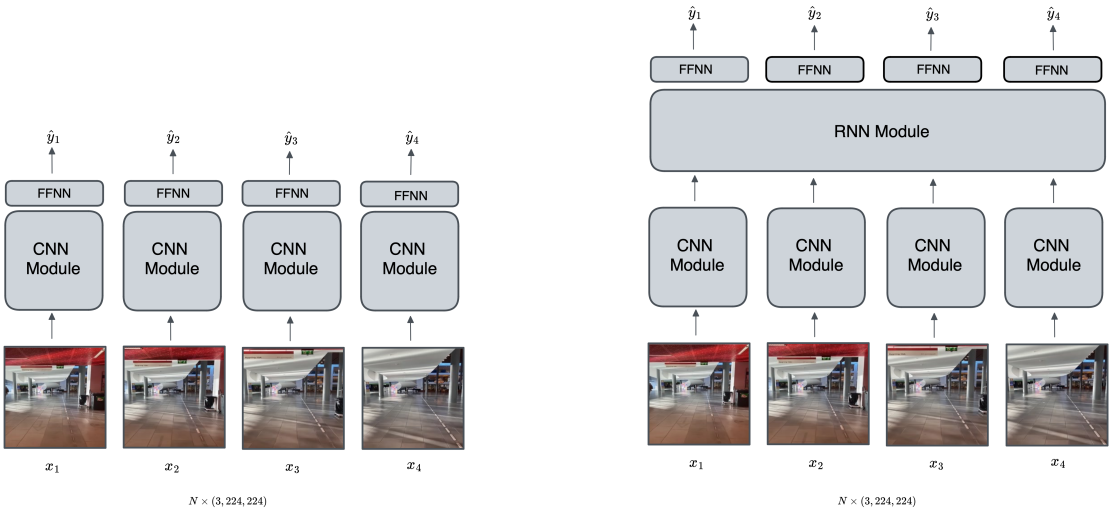



Figure 2: Examples of preprocessed frame-location pairs




(a) Image Classification Architecture

(b) Video Classification Architecture

Figure 3: Model Architectures

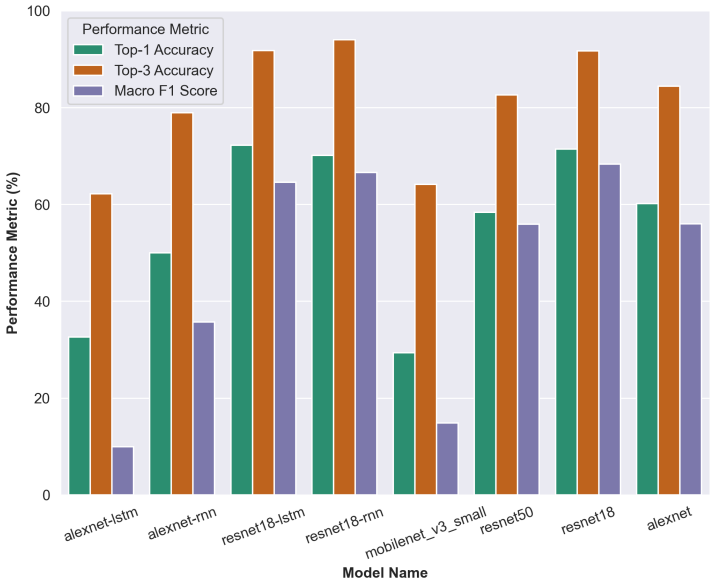



Figure 4: **Performance Metrics.** The three classification performance metrics are shown for all models in a grouped bar chart.

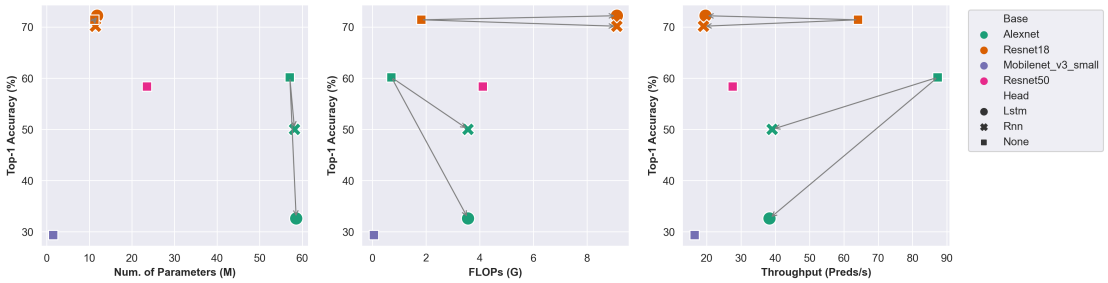



Figure 5: **Performance-Efficiency Trade-Off.** The three Figures show the Top-1 Accuracy (%) against (a) the number of parameters, (b) the number of floating point operations and (c) the throughput (predictions per second) for all models. The models are coloured according to their CNN base module and the marker style indicates the type of RNN head. The grey lines aid the visualisation of the development of performance-efficiency trade-off with different recurrent modules.

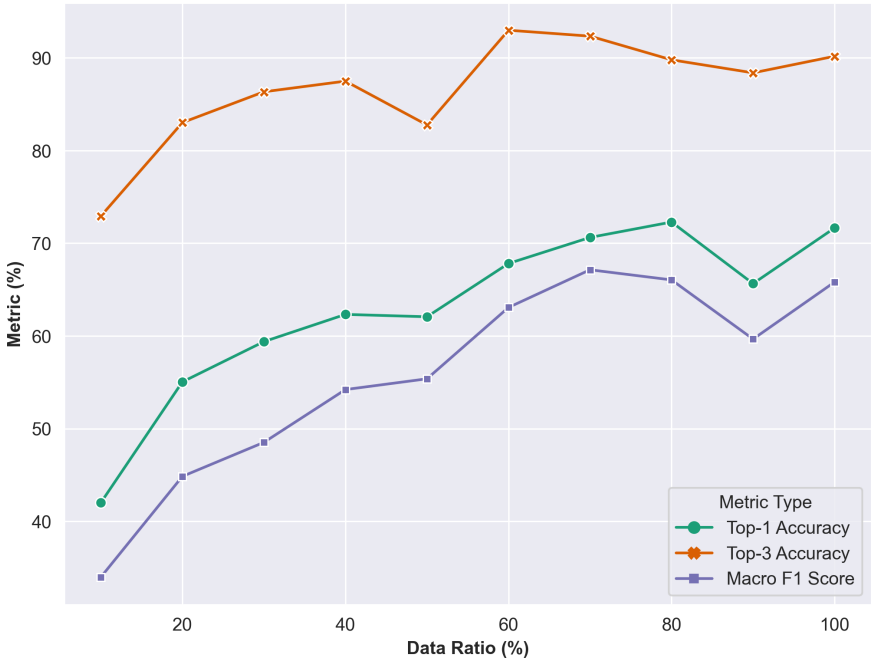
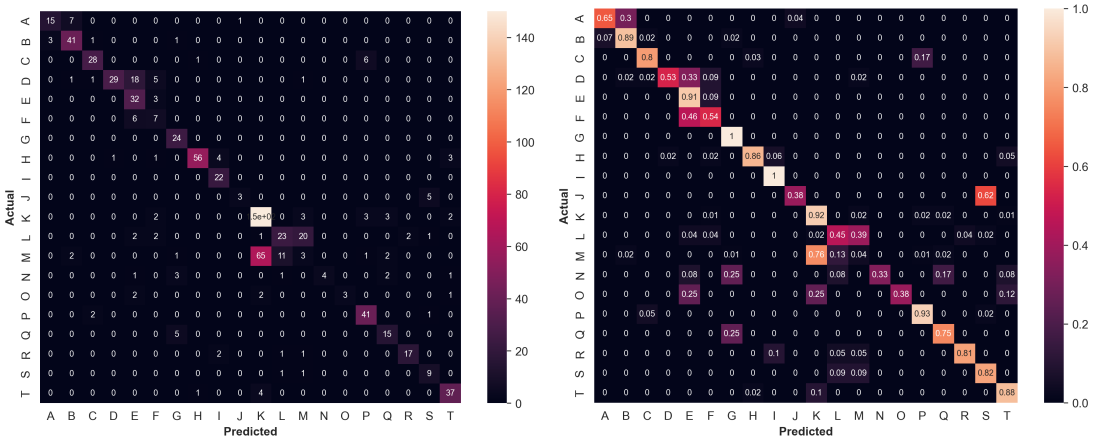


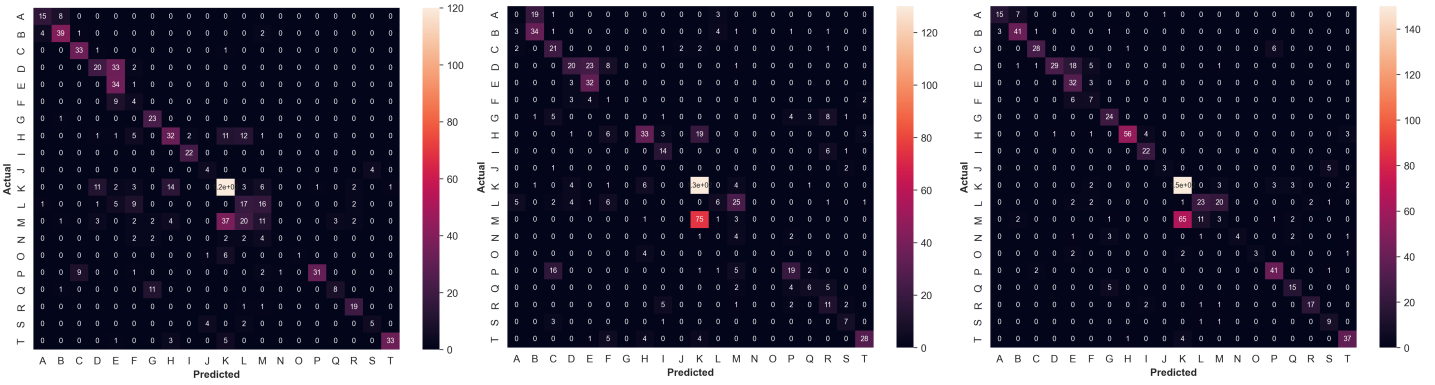

Figure 6: **Data Efficiency.** The figure shows the performance of the baseline model `resnet18` when trained on different training data metrics.



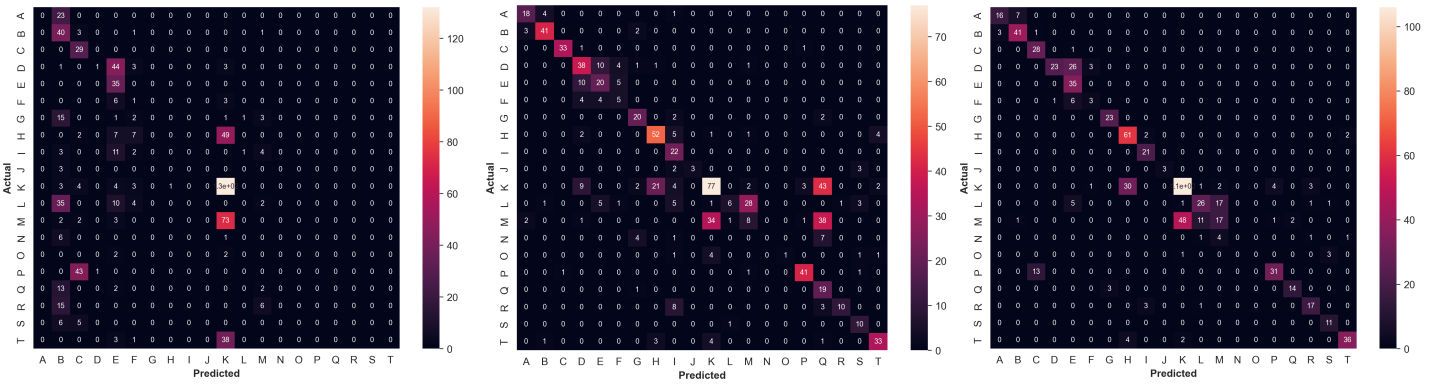

(a) Unnormalised Confusion Matrix (b) Row-Normalised Confusion Matrix

Figure 7: **ResNet18 Confusion Matrix.** The figure shows the (a) unnormalised and (b) row-normalised confusion matrix. For (a), the entry at row $i$ and column $j$ shows the number of samples that belong to class $i$ but were predicted to be class $j$. For (b), the entry at row $i$ and column $j$ shows

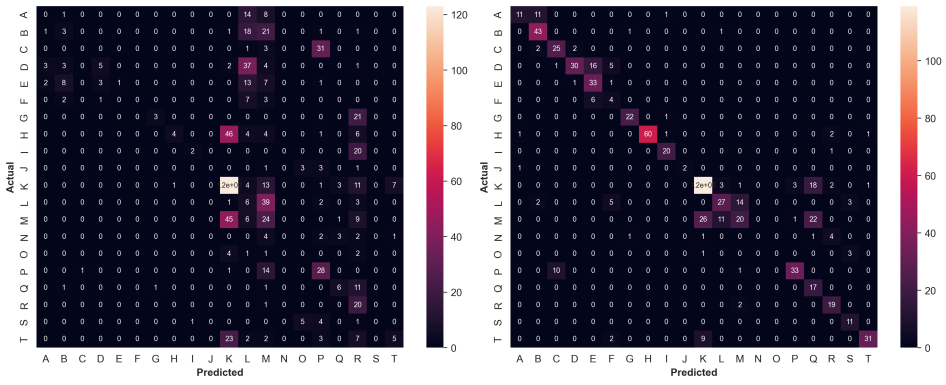
Figure 8: **PlayTorch App.** Download the PlayTorch iOS or Android app and scan the QR code to try out the model on your mobile device.




(a) `alexnet` (b) `alexnet-rnn` (c) `resnet18`

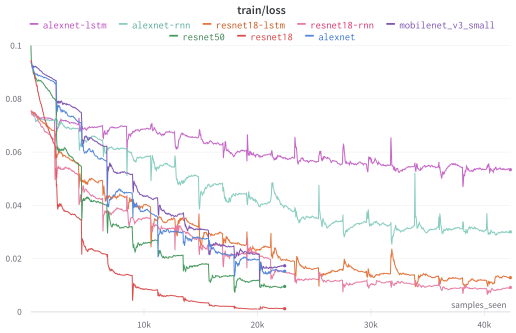



(d) `alexnet-lstm` (e) `resnet50` (f) `resnet18-rnn`




(g) `mobilenet_v3_small` (h) `resnet18-lstm`

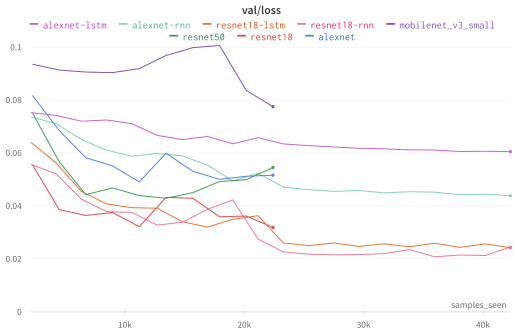
Figure 9: **Confusion Matrices.** The table shows the (unnormalised) confusion matrices of all models in Experiment 1. The predictions are computed on the frames from the test split and the matrix. The entry at index $(i, j)$ is the number of samples with true class $i$ that were predicted to be class $j$.



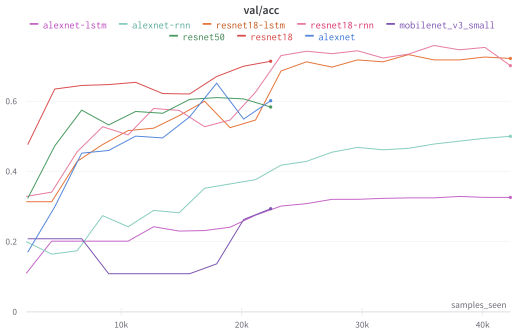

(a) Training Loss/ Samples


(b) Training Accuracy/ Samples

Figure 10: Training Metrics for Experiment 1




(a) Validation Loss/ Samples




(b) Valiation Accuracy/ Samples

Figure 11: Validation Metrics for Experiment 1

| **Class** | **Encoding** |
|---|---|
| First_Floor_Corridor_1 | A |
| First_Floor_Corridor_2 | B |
| First_Floor_Green_Area | C |
| First_Floor_Library_1 | D |
| First_Floor_Library_2 | E |
| First_Floor_Library_3 | F |
| First_Floor_Magenta_Area | G |
| First_Floor_Mezzanine | H |
| First_Floor_Red_Area | I |
| First_Floor_Yellow_Area | J |
| Ground_Floor_Atrium | K |
| Ground_Floor_Corridor_1 | L |
| Ground_Floor_Corridor_2 | M |
| Ground_Floor_Entrance_Magenta | N |
| Ground_Floor_Entrance_Yellow | O |
| Ground_Floor_Green_Area | P |
| Ground_Floor_Magenta_Area | Q |
| Ground_Floor_Red_Area | R |
| Ground_Floor_Yellow_Area | S |
| Stairs_Atrium | T |

Table 7: Encoding of Classes