DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Video classification with memory and computation-efficient convolutional neural network

BENJAMIN NAOTO CHICHE

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Video classification with memory and computation-efficient convolutional neural network

BENJAMIN NAOTO CHICHE

Master in Computer Science
Date: July 1, 2019
Supervisor: Amir H. Payberah
Examiner: Henrik Boström
School of Electrical Engineering and Computer Science
Host company: Microsoft France
Swedish title: Videoklassificering med minnes-och beräkningseffektivt convolutional neural network

Abstract

Video understanding involves problems such as video classification, which consists in labeling videos based on their contents and frames. In many real world applications such as robotics, self-driving car, augmented reality, and Internet of Things (IoT), video understanding tasks need to be carried out in a real-time manner on a device with limited memory resources and computation capabilities, while meeting latency requirement.

In this context, whereas neural networks that are memory and computation-efficient - i.e., that present a reasonable trade-off between accuracy and efficiency with respect to memory size and computational speed - have been developed for image recognition tasks, studies about video classification have not made the most of these networks. To fill this gap, this project answers the following research question: how to build video classification pipelines that are based on memory and computation-efficient convolutional neural network (CNN) and how do the latter perform?

In order to answer this question, the project builds and evaluates video classification pipelines that are new artefacts. This research involves triangulation (i.e., is qualitative and quantitative at the same time) and the empirical research method is used for the evaluation. The artefacts are based on one of existing memory and computation-efficient CNNs and its evaluation is based on a public video classification dataset and multiclass classification performance metrics. The case study research strategy is adopted: we try to generalize obtained results as far as possible to other memory and computation-efficient CNNs and video classification datasets. The abductive research approach is used in order to verify or falsify hypotheses. As results, the artefacts are built and show satisfactory performance metrics compared to baseline pipelines that are also developed in this thesis and metric values that are reported in other papers that used the same dataset. To conclude, video-classification pipelines based on memory and computation-efficient CNN can be built by designing and developing artefacts that combine approaches inspired from existing papers and new approaches and these artefacts present satisfactory performance. In particular, we observe that the drop in accuracy induced by memory and computation-efficient CNN when dealing with video frames is, to some extent, compensated by capturing temporal information via consideration of sequence of these frames.

Sammanfattning

Videoförståelse innebär problem som videoklassificering, som består av att annotera videor baserat på deras innehåll och ramar. I många verkliga applikationer, som robotteknik, självkörande bilar, förstärkt verklighet (AR) och sakernas internet (IoT) måste videoförståelsuppgifter utföras i realtid på en enhet med begränsade minnesresurser och beräkningsförmåga, samtidigt som det uppfyller krav på låg fördröjning.

I det här sammanhanget, medan neurala nätverk som är minnes- och beräkningseffektiva, dvs den aktuella presentationen har en rimlig avvägning mellan noggrannhet och effektivitet (med avseende på minnesstorlek och beräkningar) utvecklats för bildigenkänningsuppgifter, har studier om videoklassificering inte fullt utnyttjat dessa tekniker. För att fylla denna lucka i vetenskapen svarar det här projektet på följande forskningsfråga: hur bygger man videoklassificeringspipelines som bygger på minne och beräkningseffektiva faltningsnätverk (CNN) och hur utförs det sistnämnda?

För att svara på denna fråga bygger projektet och utvärderar videoklassificeringspipelines som är nya artefakter. Den empiriska forskningsmetoden används i denna forskning som involverar triangulering (dvs kvalitativt och kvantitativt samtidigt). Artefakterna är baserade på ett befintligt minnes- och beräkningseffektivt CNN och dess utvärdering baseras på en öppet tillgängligt dataset för videoklassificering. Fallstudieforskningsstrategin antas: Vi försöker att generalisera erhållna resultat så långt som möjligt till andra minnes- och beräkningseffektiva CNNs och videoklassificeringsdataset. Som resultat byggs artefakterna och visar tillfredsställande prestandamätningar jämfört med baslinjeresultat som också utvecklas i denna avhandling och värden som rapporteras i andra forskningspapper baserat på samma dataset. Sammanfattningsvis kan video-klassificeringsledningar baserade på ett minne och beräkningseffektivt CNN byggas genom att utforma och utveckla artefakter som kombinerar metoder inspirerade av befintliga papper och nya tillvägagångssätt och dessa artefakter presenterar tillfredsställande prestanda. I synnerhet observerar vi att nedgången i noggrannhet som induceras av ett minne och beräkningseffektivt CNN vid hantering av videoramar kompenseras till viss del genom att ta upp tidsmässig information genom beaktande av sekvensen av dessa ramar.

Contents

Chapter 1

Introduction

This chapter describes the specific problem that this thesis project addresses, the context of the problem, the goals of this project, and outlines the structure of the thesis.

1.1 Background

Research in image understanding - involving problems such as image classification and object detection - has known accelerated improvement. More and more complicated and deeper neural networks (NNs) intended for doing image recognition with the highest possible accuracy - widely based on Convolutional Neural Networks (CNNs) that can effectively extract features from images [1] - emerged. AlexNet [2], VGG [3], ResNet [4], and GoogleLeNet/Inception [5, 6], are examples of such networks, and proved their power based on large and diverse image datasets, such as Pascal VOC data sets [7] or ImageNet [8, 9].

However, that trend of NN models that became more and more complicated and deeper decreased their efficiency regarding size and speed. In many real world applications such as robotics, self-driving car, augmented reality, and Internet of Things (IoT), the recognition tasks need to be carried out in a real-time manner on a device with limited memory resources and computation capabilities, while meeting latency requirement. Therefore, more recently, on the one hand, more memory and computation-efficient (i.e., efficient with respect to memory size and computational speed) deep learning (DL) models have been developed for image recognition. MobileNetV1 [10], MobileNetV2 [11], SqueezeNet [12] and ShuffleNet [13, 14] are examples of such networks. They allowed to do a less but enough accurate and much faster image recognition tasks and they are lighter than the previously mentioned complicated deep neural networks (DNNs) in terms of memory. They present a reasonable trade-off between accuracy and efficiency. Note that memory size of a model and its computational speed are strongly related in DL; for example, a DL model that has less parameters has obviously smaller memory size and involves linear combinations that have less terms so that are faster. On the other hand, DL model compression and acceleration techniques, based on quantization [15, 16, 17, 18], parameter pruning and sharing, low-rank factorization, transferred/compact convolutional filters and knowledge distillation [19, 20, 21], have been studied. Newer engineering tools that benefit from these studies have been developed, such as TensorFlow Lite [22].

Besides, video understanding involves problems such as video classification, which consists in labeling videos based on their contents and frames. Action recognition is an example of video classification, which consists in labeling action videos. To a lesser degree than image classification, studies have been carried out in order to deal with video classification problem. Most of them used image features extracted from video frames based on CNNs and/or *optical flow*, which is a local spatio-temporal feature that is useful to extract motion information from succession of video frames. Image features give spatial (appearance) information and their aggregation over time gives temporal (motion) information. Optical flow directly captures motion information. [23, 24] proposed two-stream architectures that extracted spatial information from single frame of a video based on CNNs, dealt with motion information by computing optical flow and by using CNNs in order to extract features from optical flow images, and then aggregated results of these two approaches. [25] used CNNs to extract spatial features from individual video frames then combined them in order to capture temporal information, by using approaches such as Long-Short-Term-Memory (LSTM). The study also used optical flow to have another source of motion information. [26] labelled videos and extracted Inception-v3 [27] features from their frames in order to create a large and diverse video dataset and trained classifiers on it, by trying or not to integrate information over time.

1.2 Problem

The studies about video classification have not made the most of the aforementioned progress in image understanding regarding memory and computation-efficient DNNs. Building of video classification pipelines based on them deserves more attention, given the existence of numerous possible applications involving the analysis of video data at the level of devices with limited memory resources and computation capabilities. This is one of the points that motivate this thesis. The aforementioned studies leveraged feature extractions that are difficult to do in practice on these devices. The computation of optical flow is costly in terms of execution time and memory [28, 29] thus became a bottleneck in [23]. Moreover, CNNs that are used in the two-stream architectures [23, 24] that extracted features from frames and optical flow images are deep and complicated CNNs that are not designed to be memory and computation-efficient (CNN-M-2048 [30] in [23]; VGG-M-2048 and VGG-16 [30, 3] in [24]). [25] extracted features from images by using AlexNet [2] and GoogleNet [5] and [26] extracted features from images by using Inception-v3 [27]. All of these networks are deep, complicated and not meant to be efficient in terms of memory and computation. For our best knowledge, no studies about video classification have used memory and computation-efficient CNNs such as MobileNets or benefited from DL compression and acceleration techniques when dealing with image feature extraction. This is the second point that motivates this thesis. Moreover, regarding video classification based on memory and computation-efficient CNN, whereas the latter is less accurate when considering individual video frames, this might be compensated by capturing temporal information via consideration of sequence of these frames. This is the third point that motivates this thesis.

1.3 Purpose

As mentioned in the previous section, the studies about video classification have not made the most of memory and computation-efficient CNNs. This thesis aims to address this problem by building video classification pipelines that are based on memory and computation-efficient CNN. This raises the following question: how to actually build these video classification pipelines? To answer this question, as a case study, video classification pipelines that are based on quantized MobileNetV2 are built. Specificities of this network are detailed in chapter 2. The reason for its use is that, in a nutshell:

- MobileNetV2 is one of the NNs that are designed for image recognition tasks that presents the most effective trade-off between accuracy and model efficiency in terms of model size (memory size) and computational speed;
- Quantization is one of the most effective techniques that aims to reduce DL model size and computational speed

and they can be combined in order to obtain a highly memory and computation-efficient and accurate enough DL image recognition model. Quantized MobileNetV2 is chosen in this thesis as a representative of memory and computation-efficient CNNs that also include MobileNetV1, MobileNetV2, quantized MobileNetV1 and other networks. Indeed, exactly same approaches as the approaches that are developed in this thesis can be recycled when building video classification pipelines that are based on one of these memory and computation-efficient CNNs. Therefore, the thesis first aims to design and build the pipelines that are indeed new artefacts. They are inspired from existing papers, but are different from the approaches they proposed. They should capture spatial information brought by video frames and temporal information by considering sequences of video frames.

Moreover, it is important to know how well the artefacts can perform. Subsequently, this allows to be sure they correctly perform and to validate them. This raises the following question: how well do the video classification pipelines based on memory and computation-efficient CNN perform? In order to answer this question, the built video classification pipelines' performance is empirically measured and compared with other baseline pipelines that are based on a deeper and more accurate image recognition CNN (Inception-v3) and that are also built in this thesis following the same design as the pipelines that are based on quantized MobileNetV2. Here also, the use of quantized MobileNetV2 and Inception-v3 involves a case study with a view to generalizing as far as possible obtained results.

Overall, by concatenating the two questions, this thesis answers the following research question: how to build video classification pipelines that are based on memory and computation-efficient CNN and how do the latter perform?

Along with answering this question, the thesis tries to first verify or falsify the following hypothesis, based on the abductive research approach: the video classification pipelines that are based on quantized MobileNetV2 perform not as good as the baseline pipelines but they present similar performance. If this is the case, it may indicate that the drop in accuracy induced by the use of memory and computation-efficient CNN when dealing with video frames can be compensated by capturing temporal information via consideration of sequence of these frames. To support the latter, also single-frame models that completely ignore temporal information are built on top of both quantized MobileNetV2 and Inception-v3 and evaluated. Moreover, performance metrics is measured based on an appropriate video classification dataset that is collected. This dataset should serve as a representative of video classification datasets, with a view to again generalizing our results to other video classification datasets.

The purpose of this thesis is not to achieve a state-of-the-art result based on the dataset. Instead, based on these metrics and the abductive research approach, the second hypothesis that can be verified or falsified is that, regarding the dataset, the video classification pipelines based on quantized MobileNetV2 perform not as good as the state-of-the-art results presented in other papers [23, 24, 25] based on very deep and complicated DNNs and/or optical flow, but enough good to be considered acceptable.

1.4 Goal

As long-term goal, the completion of this work is a significant starting point for video classification pipelines that are based on a bit less accurate but memory and computation-efficient CNNs that can differ or not from quantized MobileNetV2. As stated above, quantized MobileNetV2 is chosen as representative of memory and computation-efficient CNNs that also include MobileNetV1, quantized MobileNetV1, MobileNetV2 and other networks. Measures of performance metrics, along with comparison with the reference (baseline) pipelines, allows to have an idea about how well video classification based on these networks can perform and validate them. In the same perspective, the hypothesis about the drop in accuracy induced by memory and computation-efficient CNN at video frame level that can be compensated by capturing temporal information via consideration of sequence of these frames is an interesting point to discuss. This thesis can show potential utility of memory and computation-efficient CNN when dealing with video classification, which is interesting for one who wants to do video classification on a platform that is limited in terms of memory and computation capabilities. The existence of numerous possible applications involving the analysis of video data at the level of devices with limited memory resources and computation capabilities strengthens the importance of this work.

1.5 Methodology

Research method for this degree project is determined following the portal presented in [31]. First of all, the positivism philosophical assumption is made, i.e., the reality is objectively given and independent of the observer and instruments. We conduct a qualitative and quantitative research, which involves triangulation. Indeed, to answer the question, video classification pipelines which are indeed new artefacts are designed and numerically evaluated. This is a qualitative research because this involves development of new artefacts; this is also a quantitative research because this involves measurement of numerical metrics. To answer the question, video classification pipelines that are based on quantized MobileNetV2 that is chosen as a representative of memory and computation-efficient CNNs are designed, implemented and empirically evaluated based on the empirical research method. Indeed, exactly same approach can be recycled when building video classification pipelines that are based on other memory and computation-efficient CNNs, which allows generalization of the conclusions from this thesis. Therefore here, we adopt the case study research strategy. Performance metrics that are related to multiclass classification task are used for the evaluation. To have a reference (baseline) for comparison, similar video classification pipelines, but this time based on Inception-v3 are also developed and evaluated. The abductive research approach is used in order to verify or falsify the hypotheses. Conclusions are drawn from experimental results and observations based on a dataset. The experiment and case study data collection is done to obtain the dataset. The computational mathematics method is used with a view to analyzing this dataset. Again, the pipelines in this study is based on quantized MobileNetV2 and the dataset, but we try to generalize as far as possible the results to other memory and computation-efficient CNNs and other video classification datasets.

1.6 Delimitations

Some aspects of the research question will not be considered. Regarding the first part of it, there are many possibilities when building video classification pipelines based on a CNN, even without dealing with optical flow. [26] proposes three frame-level features-based models and three video-level features-based models (those video-level features are obtained from frame-level features). [25] proposes several feature pooling architectures and a LSTM architecture that can use optical flow. [32] proposes four fusion methods. This thesis does not deal with all of them, because of time constraint. Instead, it designs video classification pipelines that are indeed new artefacts that are different from but highly inspired from the LSTM-based approaches that are presented in [26, 25].

Regarding the second part of the research question, "performance" is an ambiguous term, because for a classification pipeline this can concern several different aspects such as memory complexity, time complexity end energy consumption. In this study only the performance metrics related to the multiclass classification tasks are measured. Indeed, it is obvious that the pipelines based on memory and computation-efficient CNN improve performance in terms of memory size, energy consumption and computational speed.

Quantized MobileNetV2 is chosen in the case study research strategy as a representative of memory and computation-efficient CNNs that also include MobileNetV1, quantized MobileNetV1, MobileNetV2 and other networks. We will only build video classification pipelines based on quantized MobileNetV2 and the reference pipelines but conclusions are generalised as far as possible to other memory and computation-efficient CNNs. The same statement stands for datasets: conclusions drawn from observations that are based on the selected and used dataset are generalized as far as possible to other video classification datasets.

Finally, as already mentioned, the purpose of this thesis is not to achieve a state-of-the-art result based on the dataset. Instead, a hypothesis that can be verified or falsified is that, regarding the dataset, the video classification pipelines perform not as good as the state-of-the-art results presented in other papers based on very deep and complicated DNNs and/or optical flow, but enough good to be considered acceptable.

1.7 Benefits, Ethics and Sustainability

As previously mentioned, this work can serve as a significant starting point for video classification pipelines that are based on memory and computation-efficient CNNs. Validation or non-validation of the hypothesis about the drop in accuracy induced by memory and computation-efficient CNN at video frame level that can be compensated by capturing temporal information via consideration of sequence of these frames is an interesting point from this perspective. This thesis can show utility of memory and computation-efficient CNN when dealing with video classification, which is interesting for one who wants to do video classification on a platform that is limited in terms of memory and computation capabilities.

However, the deployment of such video classification can violate one's privacy, by allowing him to be monitored without his consent. From this point of view, the video classification and its mechanism can be abusively used or modified for malicious intention. This project has been proposed by the host company in order to only satisfy academic requirement. It is based on a public for research and benchmark dataset, that is collected in an ethical way.

Besides, the use of memory and computation-efficient CNN when dealing with video classification can significantly reduce power consumption, as these CNNs require less memory and faster computations. The quantization technique allows to do computations on reduced number of bits. In this sense, the artefacts that are designed and developed in this thesis are sustainable.

1.8 Outline

Chapter 2 presents extended background. Chapter 3 details the research method that is used in this project. Chapter 4 describes and discusses obtained results. Chapter 5 concludes this thesis work and provides insight for future work.

Chapter 2

Background

This chapter provides theoretical background of different concepts that need to be understood by readers of this thesis.

2.1 Artificial Neural Networks and Deep Neural Networks

Artificial neural networks (ANN) are computing systems that are inspired by the biological neural networks of brains. It is based on a set of connected units or neurons. Each connection, similarly to the synapses in a biological brain, transmits a signal from one neuron to another. A neuron that receives a signal processes it and then sends signal to other neurons connected to it. In ANN implementations, the signal is a real number, and the output of each neuron is computed by applying non-linear function of the weighted sum of its inputs. Indeed, neurons and connections between neurons (edges) have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that the signal is only sent if the weighted sum crosses that threshold. Generally, neurons are aggregated into layers. Different layers perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. A DNN is an ANN with multiple layers between the input and output layers.

In the supervised machine learning context - given data points and their associated outputs -, the goal of DL for a DNN is to learn the weights of its edges, based on training.

The first six subsections in the following are highly inspired from [33].

2.1.1 Feedforward neural network

A Feedforward neural network (FNN) is an artificial neural network in which connections do not form a cycle. The latter is composed of:

- One input layer
- One or more hidden layers
- One final output layer

Every layer except the output layer includes a bias neuron and is fully connected to the next layer. The model is associated with a directed acyclic graph describing how the functions are composed together. The length of the chain gives the depth of the model. Figure 2.1 illustrates a FNN. Neurons may have a threshold such that the signal is only sent if the weighted sum crosses that threshold, as stated above. However, in most cases, they are replaced by activation functions. Let x be the weighted sum which is an input for a neuron, its output is $y = activation(x)$. Examples activation functions are:

- $sigmoid(x) = \frac{1}{1+e^{(-x)}}$.
- $tanh(x) = \frac{e^{(x-)}-e^{(-x)}}{e^{(x)}+e^{(-x)}}$.
- $ReLU(x) = max(x, 0)$.
- $LeakyReLU_{\alpha}(x) = max(\alpha x, x)$, with α being a parameter.

Each activation function has its advantages and drawbacks.

Training of FNN is based on the backpropagation algorithm combined with an optimisation algorithm (2.1.4).

2.1.2 Convolutional neural network

Convolutional neural network (CNN) is a type of deep neural networks, that is mostly used in visual image analysis. They were inspired by biological processes in the sense that the connectivity between neurons is similar to the way the animal visual cortex works. Individual neurons respond to stimuli only in a restricted region of the visual field, called the receptive field. The receptive fields of different neurons partially overlap so that they cover the entire visual field.

Indeed, tackling image understanding based on FNN faces difficulty in recognizing objects, due to phenomena such as:

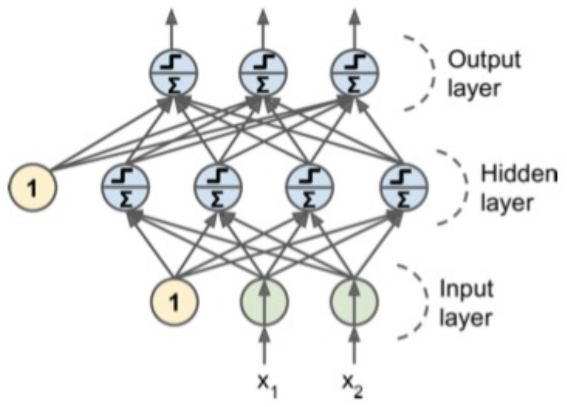


Figure 2.1: An illustration of a FNN [33]

- Rotation
- Lighting: objects may look different depending on the level of external lighting.
- Deformation: objects can be deformed in a variety of non-affine ways.
- Scale variation: visual classes often exhibit variation in their size.
- Viewpoint invariance.

A CNN can tackle these challenges, taking advantage of shape information. It is composed of a stack of convolutional modules. Each module consists of a convolutional layer followed by a pooling layer:

- Each neuron in a convolutional layer applies filters on its receptive field: calculates a weighted sum of the input pixels in the receptive fields, adds a bias, and feeds the result through its activation function to the next layer. Figure 2.2 illustrates this filtering operation. The amount of movement between applications of the filter to the input image is called stride, and it is generally symmetrical in height and width dimensions. The addition of pixels to the edge of the image is called padding. The output of this layer are feature maps (activation map). As input images are also composed of multiple sub layers - one per color channel -, a convolutional layer simultaneously applies multiple filters to its inputs.
- The pooling layer downsamples the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time.

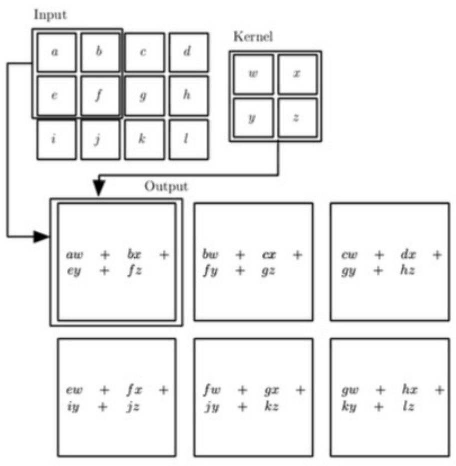


Figure 2.2: An illustration of the filtering operation [33]

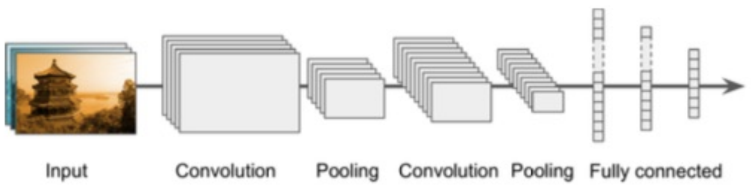


Figure 2.3: An illustration of the CNN architecture [33]

The last module is followed by the flattening operation and one or more dense layers that perform classification. Flattening converts the output of the convolutional part of the CNN into a 1D feature vector. The final dense layer contains a single node for each target class in the model, with a softmax activation function. Figure 2.3 illustrates the architecture of the CNN. Its training is also based on the backpropagation algorithm (2.1.4).

2.1.3 Recurrent neural network

The idea behind Recurrent neural networks (RNN) is to deal with sequential data: inputs (and outputs) are not independent of each other. Neurons in an RNN have connections pointing backward, and RNNs have memory, which captures information about past computations. Figure 2.4 illustrates a RNN neuron. Each recurrent neuron has three sets of weights: u, w, and v.

- u: the weights for the inputs $x^{(t)}$.

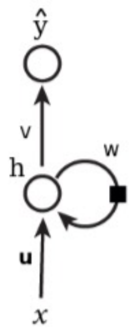


Figure 2.4: An illustration of a RNN neuron [33]

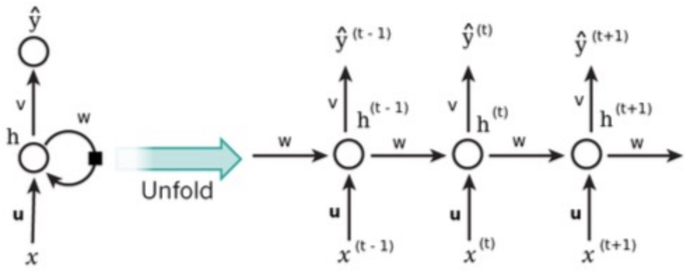


Figure 2.5: An illustration of the RNN unfolding [33]

- w: the weights for the hidden state of the previous time step $h^{(t-1)}$.
- v: the weights for the hidden state of the current time step $h^{(t)}$.

To train an RNN, one should unroll it through time and then do the back-propagation (2.1.4). This is called backpropagation through time. Figure 2.5 illustrates the network unrolling or unfolding.

By stacking multiple layers of cells, a deep RNN is obtained.

LSTM

RNNs presents problems:

- when the gap between the relevant information and the place that it's needed grows, RNNs become unable to learn to connect the information.
- RNNs may suffer from the vanishing/exploding gradient problem (see 2.1.5 for its definition).

Long short-term memory (LSTM) [34] has been introduced in order to solve these problems. In LSTM, the network can learn what to store and what to throw away. Figure 2.6 illustrates the structure of a LSTM cell. Without looking inside the box, the LSTM cell looks exactly like a basic cell. Whereas the repeating module in a standard RNN contains a single layer, the repeating module in an LSTM contains four interacting layers. In LSTM state is split in two vectors:

- $h^{(t)}$ (h stands for hidden): the short-term state
- $c^{(t)}$ (c stands for cell): the long-term state. The LSTM can remove/add information to the cell state, regulated by three gates: forget gate, input gate and output gate.

The following steps summarize the LSTM Walk:

- A sigmoid layer, called The forget gate layer, decides what information we are going to throw away from the cell state. It looks at $h^{(t-1)}$ and $x^{(t)}$, and outputs a number between 0 and 1 for each number in the cell state $c^{(t-1)}$. 1 represents completely keep this, and 0 represents completely get rid of this. We have: $f^{(t)} = \sigma({u_f}^T x^{(t)} + w_f h^{(t-1)})$
- A sigmoid layer, called the input gate layer, decides which values we will update. A tanh layer creates a vector of new candidate values that could be added to the state. We have: $i^{(t)} = \sigma({u_i}^T x^{(t)} + w_i h^{(t-1)})$ and $\tilde{c}^{(t)} = tanh(u_{\tilde{c}}^T x^{(t)} + w_i h^{(t-1)})$. These the two layers overall decide what new information we are going to store in the cell state.
- The old cell state $c^{(t-1)}$ is updated into the new cell state $c^{(t)}$. We multiply the old state by $f^{(t)}$, forgetting the things we decided to forget earlier. Then we add it $i^{(t)} \bigotimes c^{(t)}$. This is the new candidate. values, scaled by how much we decided to update each state value. We have that $c^{(t)} = f^{(t)} c^{(t-1)} + i^{(t)} c^{\ (t)}$
- The final step is the decision about the output. First, a sigmoid layer decides what parts of the cell state we are going to output. Then, the cell state is put through tanh and multiplied by the output of the sigmoid gate (outputgate), so that it only outputs the parts it decided to. We have that $o^{(t)} = \sigma({u_o}^T x^{(t)} + w_o h^{(t-1)})$ and $\hat{y}^{(t)} = h^{(t)} = o^{(t)} \bigotimes tanh(c^{(t)})$.

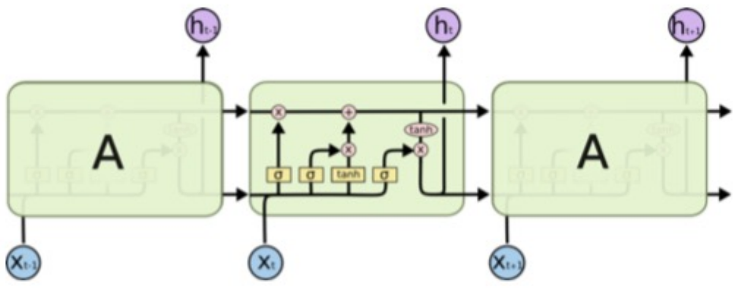


Figure 2.6: An illustration of a LSTM cell [33]

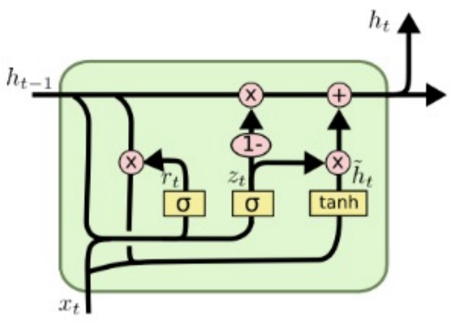


Figure 2.7: An illustration of the GRU cell [36]

GRU

A variation on the LSTM is the Gated Recurrent Unit (GRU), introduced by [35]. It combines the forget and input gates into a single update gate. It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular. Figure 2.7 illustrates this cell. We have that:

- $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$
- $tanh(W \cdot [r_t \cdot h_{t-1}, x_t])$
- $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$

2.1.4 Training DNNs

In the context of multiclass classification, with y being the ground truth label and $\hat{y}$ being the predicted label, the cost function (or the loss function) is the cross-entropy between them:

$$J(w) = -\frac{1}{m}\sum_i \sum_j y_j^{(i)} log(\hat{y}_j^{(i)})$$

This quantity quantifies the difference (error) between two probability distributions. It is also mathematically equivalent to the negative log-likelihood in probability theory.

The goal of DL is to find w that minimizes $J(w)$. To do so, the basic method to use is the gradient decent. Starting from a random point w_0, one repeats the following steps, until the stopping criterion is satisfied :

- Determine a descent direction $\frac{\delta J(w)}{\delta w}$.
- Choose a step size η.
- Update the parameters: $w_i^{(next)} = w_i - \eta \frac{\delta J(w)}{\delta w_i}$ (simultaneously for all parameters).

In DL, the computation of $\frac{\delta J(w)}{\delta w}$ is based on the backpropagation training algorithm. For each training instance $x^{(i)}$ the algorithm does the following steps:

- Forward pass: make a prediction (compute $\hat{y}^{(i)} = f(x^{(i)})$).
- Measure the error (compute the cost function $cost(\hat{y}^{(i)}, y^{(i)})$). In a classification problem, the cost function is usually the cross-entropy.
- Backward pass: go through each layer in reverse to measure the error contribution from each connection.
- Tweak the connection weights to reduce the error (update the set of weights W and the set of bias b) by calculating gradients. This last step is the gradient descent step on all the connection weights in the network, using the error gradients measured earlier.

2.1.5 Techniques used in training

There are several techniques that can be used during the training in order to make the convergence faster, overcome overfitting and avoid vanishing/exploding gradient. Overfitting happens when norm of weights become too large, which equivalently shows that the model is getting too complex. Vanishing/exploding gradient happens when norm of gradient respectively becomes too small/large when doing backpropagation at lower layers (those that are closer to the inputs).

Early stopping

Early stopping is a technique used to avoid overfitting. It is based on the dataset splitting into training, validation and test sets. As the training steps go by, its prediction error on the training/validation set naturally goes down. After a while the validation error stops decreasing and starts to go back up. This shows that the model has started to overfit the training data. In the early stopping, we stop training when the validation error reaches a minimum.

Batch normalization

Batch normalization [37] is a technique to address the problem that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. It makes the learning of layers in the network more independent of each other. The technique consists of adding an operation in the model just before the activation function of each layer. It's zero-centering and normalizing the inputs, then scaling and shifting the result. It first estimates the inputs' mean and standard deviation of the current mini-batch, then uses these estimated values in order to produce scaled and shifted version of the inputs [37]. It is used in order to overcome the vanishing gradient problem: indeed, the gradient traditionally tells how to update each parameter, under the assumption that the other layers do not change, whereas in practice, we update all of the layers simultaneously, and unexpected results can happen. The technique prevents the latter.

Dropout

Dropout [38] is a regularization technique used to avoid overfitting. At each training step, each neuron drops out temporarily with a probability p:

- The hyperparameter p is called the dropout rate.

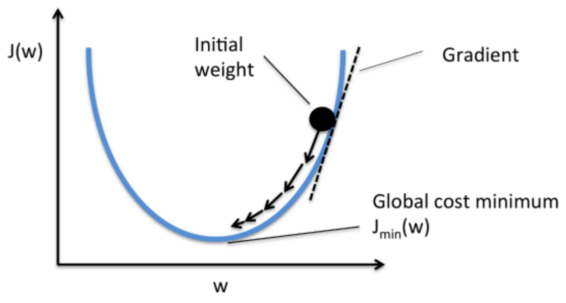


Figure 2.8: An illustration of momentum [33]

- A neuron will be entirely ignored during this training step.
- It may be active during the next step.
- Exclude the output neurons.

After training, neurons don't get dropped anymore .

Adam optimisation

Adam optimisation is an optimisation algorithm that combines the ideas of Momentum optimization and RMSProp. Like Momentum optimization,it keeps track of an exponentially decaying average of past gradients. Like RMSProp, it keeps track of an exponentially decaying average of past squared gradients.

Momentum is a concept from physics: an object in motion will have a tendency to keep moving. It measures the resistance to change in motion. The higher momentum an object has, the harder it is to stop it. This is the very simple idea behind momentum optimization(Figure 2.8):

- we can see the change in the parameters w as motion: $w_i^{(next)} = w_i - \eta\frac{\delta J(w)}{\delta w_i}$.
- we can thus use the concept of momentum to give the update process a tendency to keep moving in the same direction.
- it can help to escape from local minimums.

Momentum optimization cares about what previous gradients were.At each iteration, it adds the local gradient to the momentum vector m: $m_i = \beta m_i + \eta\frac{\delta J(w)}{\delta w_i}$ with β being called momentum and being between 0 and 1. One then updates the weights by subtracting this momentum vector: $w_i^{(next)} = w_i - m_i$

Besides, to further have a better optimization algorithm, one can keeps track of a learning rate for each parameter, and adapts the learning rate over time. To do so, parameters with large partial derivative of the cost have a rapid decrease in their learning rate and parameters with small partial derivatives have a small decrease in their learning rate. However, if the learning rate gets scaled down so much that the algorithm ends up stopping entirely before reaching the global optimum.So one can only accumulate the gradients from the most recent iterations (not from the beginning of training). This is the idea behind RMSProp.

By combining these two ideas, Adam Optimization does the following:

- $m^{(next)} = \beta_1 m + (1 - \beta_1)\Delta_w J(w)$
- $s^{(next)} = \beta_2 s + (1 - \beta_2)\Delta_w J(w) \bigotimes \Delta_w J(w)$
- $m^{(next)} = \frac{m}{1-\beta_1^T}$
- $s^{(next)} = \frac{s}{1-\beta_2^T}$
- $w^{(next)} = w - \eta m \oslash \sqrt{s + \epsilon}$

$\bigotimes$ and $\oslash$ represents the element-wise multiplication and division. Steps 1, 2, and 5 are similar to both Momentum optimization and RMSProp. In steps 3 and 4, since m and s are initialized at 0, they will be biased toward 0 at the beginning of training, so these two steps will help boost m and s at the beginning of training.

2.1.6 Quantization of NNs

Quantization of DNNs is the reduction of precision representations of weights and/or activations for both storage and computation [22]. Whereas real values are usually represented by 32-bit floats in most deep learning frameworks, thanks to quantization, DNNs can work with smaller data types with less precision, such as 8-bit integers [15]. Advantages of this are:

- Arithmetic with lower bit-depth is faster. In general, operations with 32-bit floats are slower than 8-bit integers.
- Going from 32-bits to 8-bits, almost represents a $4\times$ reduction in terms of memory.

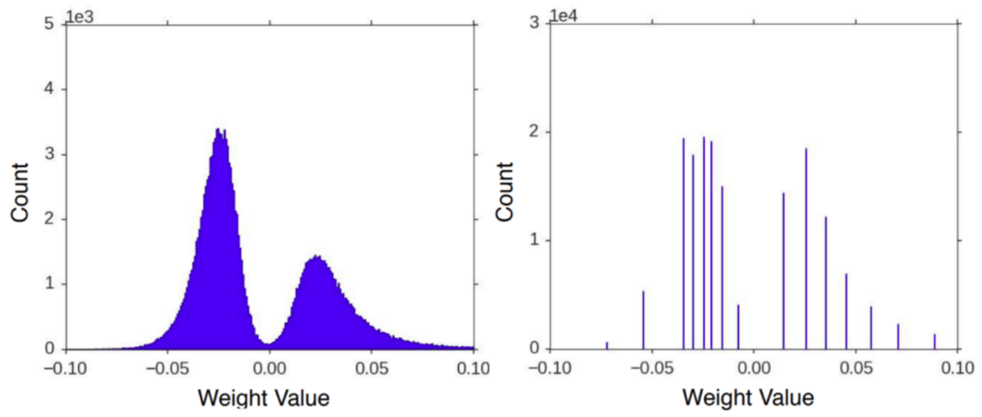


Figure 2.9: Weights in a layer from AlexNet. The right sub-graph shows one quantization using 4-bits (16 discrete values) [39]

- Lower bit-widths allow data to be stocked into the same caches or registers. This reduces number of accesses from RAM, which are costly in terms of time and power.
- Float arithmetic is hard and is not always supported on some devices, whereas integer arithmetic is readily supported.

Quantization is based on the fact that the weights and activations related to a layer generally belong to a small interval, which can be estimated in advance. This allows to concentrate fewer bits within a smaller interval. To illustrate this, Figure 2.9 shows the distribution of the weights in a layer from AlexNet [2], with a histogram of actual weights on the left [39]. One can quantize the interval to only represent some of these values accurately, and round the remaining values.

There are several levels of quantization:

- Post-training quantization: quantizes weights and activations post training, following a quantization scheme.
- Quantization-aware training: simulates quantization effects in the forward pass of training as it will happen in the inference engine, by implementing the rounding behavior of the quantization scheme. Backpropagation still happens as usual.

The following describes the 8-bit fixed point quantization scheme and quantization-aware training presented in [15] and implemented in TensorFlow Lite [22].

The latter study provided a quantization scheme that quantizes both weights and activations as 8-bit integers, and bias vectors as 32-bit integers. The latter established, by relying on two quantization parameters, the affine mapping between the bit representation of integer values q (quantized values) and their interpretation as mathematical real numbers r in order to allow efficient implementation of all arithmetic using only integer arithmetic operations on the quantized values. This scheme make use of a single set of the two quantization parameters for all values within each activations array and within each weights array. Different arrays use separate quantization parameters. This quantization scheme follows the affine equation:

$$r = \frac{r_{max} - r_{min}}{((2^B - 1)) - 0} \times (q - z) = S \times (q - z)$$

with r being the real value (generally `float32`), q being its quantized representation as a B-bit integer (for example `uint8` or `uint32`), S (`float32`) and z (`uint`) being the factors by which we scale and shift. z is the quantized 'zero-point' which will always map back exactly to 0.f (see the following). This quantization scheme satisfies the following:

- It is affine, therefore the result of fixed-point calculations can map back to real numbers.
- It always represents 0.f accurately. If we quantize and dequantize any real value, only 256 (or generally, 2^B) of them will return the exact the same number, while all others will be subject to precision loss. If we ensure that 0.f is one of these 256 values, it turns out that DNNs can be quantized more accurately according to [15] that claims that this is because 0 has a special significance in DNNs, such as padding. Besides, having 0 map to another value that is higher or lower than zero will cause a bias in the quantization scheme.

Because the weights of a pre-trained network are constant, they can be converted and stored in quantized form in advance, with their exact ranges known.

The input to a layer - or equivalently the output of a preceding layer - are also quantized with their own different parameters. While we ideally want to know the exact range of values to accurately quantize them, results of unknown inputs can still be expected to be in similar interval. We can find the average output interval on a large number of training examples and use this as a proxy to the output quantization parameters.

Regarding the function that computes the output of the layer, the results of integer computations can overflow. Therefore results have to be stored in larger integers (for example int32) and then requantized to the 8-bit output. Some of the layers' logic are changed: for example, the ReLU activation function compares values against Quantized(0) instead of 0.f. As TensorFlow Lite [22] uses gemmlowp (a type of Low-precision matrix multiplication) for matrix multiplication, which stores results of `uint8` matrix multiplications in int32, the biases are quantized in higher precision, as int32. In going from 32-bit to 8-bit, the expected quantization range is specified after the next activation layer. This will implicitly compute activations and also help the use the full quantization range in this layer.

To allow fore quantization-aware training, TensorFlow Lite introduced the "fake quantization" nodes. First, with the fake quantization nodes, the rounding effect of quantization is simulated in the forward pass of the training as it would occur in actual inference. All quantities are still stored as float during training, and backpropagation still works as usual. Second, fake quantization nodes record the ranges of activations during training. These nodes are placed in the training graph to exactly match wherever activations would change quantization ranges (input and output in Figure 2.10). As the network trains, they collect a moving average of the ranges of float values seen at that node. This is quantization-aware training.

All this information is then taken by TensorFlow Lite's TOCO (TensorFlow Optimizing Converter) tool, which, along with other optimizations, converts a neural network to the quantized form and specifies how to use them in inference by TensorFlow Lite's kernels.

2.2 CNN architectures

This section describes the families of NNs the two networks that are used in this project belong to. It also explains how they are used, based on deep transfer learning.

2.2.1 Inception-v3

GoogLeNet [5] was the winner of the ImageNet Large Scale Visual Recognition Competition in 2014 [40]. It is also called Inception-v1, and there are v2, v3 and v4 later on. In GoogLeNet, 1×1 convolution is used as a dimension reduction module to reduce the computation bottleneck, so that depth and width can be increased. This technique was introduced in [41], and is used

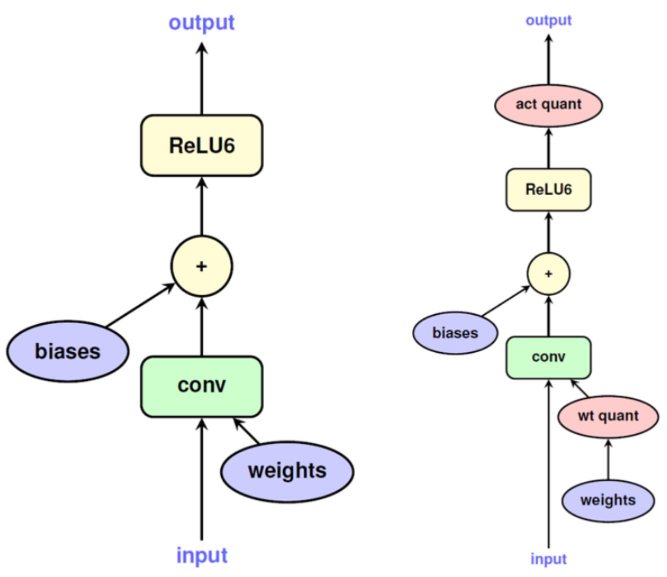


Figure 2.10: Training with simulated quantization. Left: original graph. Right: modified graph for quantization-aware training [15].

with the ReLU activation function. This dimension reduction is illustrated in Figures 2.11 and 2.12. Main building blocks of GoogLeNet is the Inception module that is based on is this dimension reduction (Figure 2.14). Global average pooling (see 2.2.3) is used nearly at the end of network by averaging each feature map from 7×7 to 1×1, as in Figure 2.22. The overall architecture is shown in Figure 2.15. There are 22 layers in total. The intermediate softmax branches are auxiliary classifiers that are only used at training time in order to overcome gradient vanishing problem, along with regularization [5].

Inception-v3 is the improvement of Inception-v2 which is in turn the improvement of GoogLeNet. Both of them were presented in the same paper [27].

Inception-v2 further reduced computational cost. 5×5 convolutions are factorized into two 3×3 convolutions. Convolutions of filter size $n \times n$ are factorized to a combination of $1 \times n$ and $n \times 1$. Then, to remove the representational bottleneck, filter banks in the module are expanded (the module is made wider instead of deeper).

To improve Inception-v2 without drastically changing the modules, Inception-v3 incorporated all of the above features for Inception-v2, and in addition used the following approaches:

- The use of RMSProp Optimizer.
- Factorization of 7x7 convolutions.

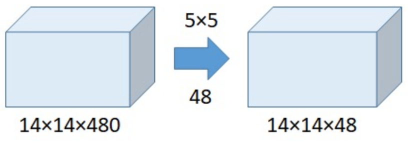


Figure 2.11: Without 1×1 convolution

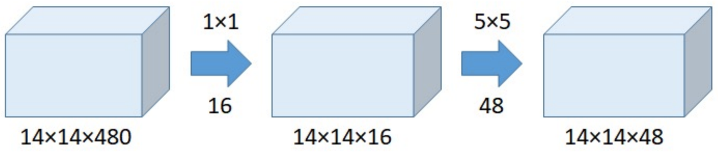


Figure 2.12: With 1×1 convolution

Figure 2.13: 5×5 convolution with the use of 1×1 convolution. Without the Use of 1×1 Convolution, number of operations = $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112.9M$. With the use of 1×1 convolution: number of operations = Number of operations for 1×1 + Number of operations for 5×5 = $(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) = 1.5M + 3.8M = 5.3M << 112.9M$ [42].

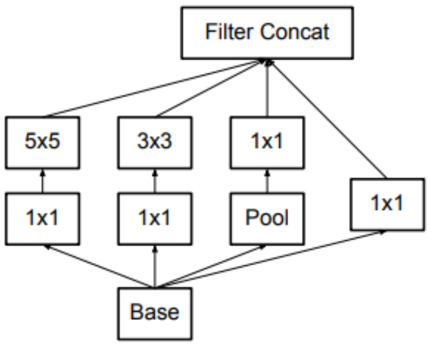


Figure 2.14: Original Inception module with the dimension reduction based on 1×1 convolution [5]

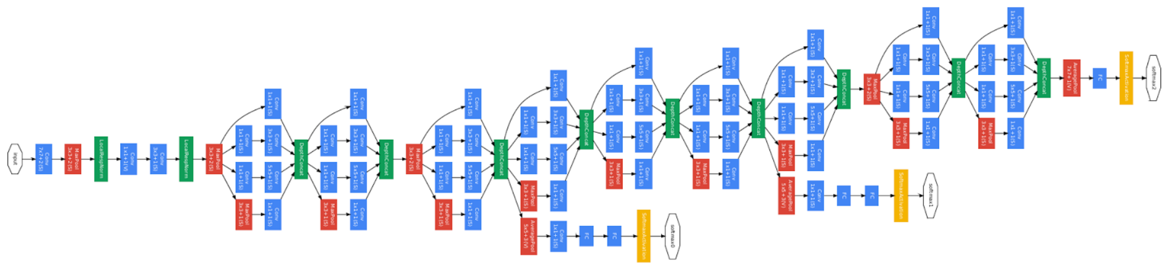


Figure 2.15: The architecture of GoogLeNet [5]

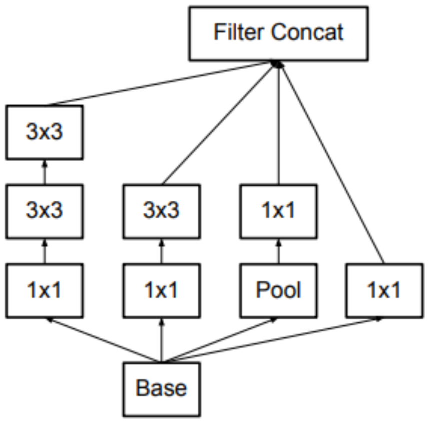


Figure 2.16: Inception module type 1 [27]

- The use of batch normalization in the auxiliary Classifiers.
- The use of label smoothing (a regularizing component that is added to the loss function in order to prevent the network from becoming too confident about a class, which prevents overfitting).

These approaches resulted in three different types of inception modules. These modules are represented in Figures 2.16, 2.17 and 2.18 and the overall architecture of Inception-v3 is presented in Table 2.1.

2.2.2 MobileNets

MobileNets regroup image understanding deep learning models that make effective trade-off between accuracy and efficiency in terms of memory and computation. Two versions of them were successively proposed [10, 11].

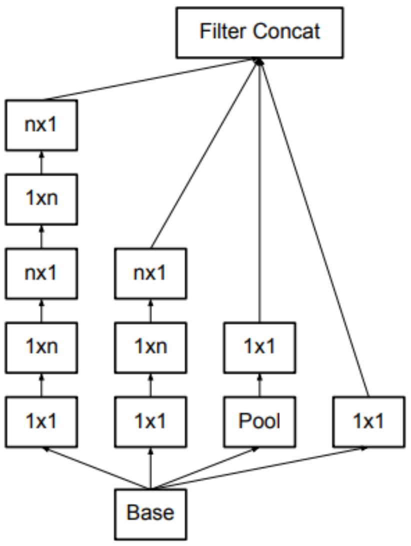


Figure 2.17: Inception module type 2 [27]

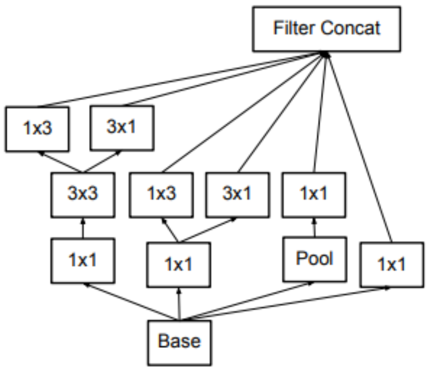


Figure 2.18: Inception module type 3 [27]

type	patch size/stride	input size
conv	$3 \times 3/2$	$299 \times 299 \times 3$
conv	$3 \times 3/1$	$149 \times 149 \times 32$
conv padded	$3 \times 3/1$	$147 \times 147 \times 32$
pool	$3 \times 3/2$	$147 \times 147 \times 64$
conv	$3 \times 3/1$	$73 \times 73 \times 64$
conv	$3 \times 3/2$	$71 \times 71 \times 80$
conv	$3 \times 3/1$	$35 \times 35 \times 192$
$3\times$ Inception	as in Figure 2.16	$35 \times 35 \times 288$
$5\times$ Inception	as in Figure 2.17	$17 \times 17 \times 768$
$2\times$ Inception	as in Figure 2.18	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Table 2.1: The outline of the Inception-v3 architecture [27]. The output size of each module is the input size of the next one.

MobileNetV1

Essential building block of MobileNetV1 is the depthwise separable convolution, which is the factorization of a standard convolution into a depthwise convolution and a pointwise convolution [10]. This reduces computation and number of parameters. It does approximately the traditional convolution operation, but much faster. Let say a standard convolutional layer's input is a $D_F \times D_F \times M$ feature map F and its output is a $D_F \times D_F \times N$ feature map G with D_F being the spatial width and height of a square input feature map, M being the number of input channels, D_G being the spatial width and height of a square output feature map and N being the number of output channel. Here it is assumed that F and G have the same spatial dimensions as the input and both of them are square [10]. The standard convolution involves parameters of convolution kernel K of size $D_k \times D_k \times N \times M$ with D_K being the spatial dimension of the kernel (assuming it is square). If stride one and padding are assumed:

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} F_{k+i-1,l+j-1,m}$$

and the standard convolution's computational cost is:

$$D_k.D_k.M.N.D_F.D_F$$

.

Depthwise convolution applies a single filter per each input channel and pointwise convolution is a 1×1 convolution. Depthwise convolution with one filter per input channel is written as:

$$\hat{G}_{k,l,n} = \sum_{i,j} \hat{K}_{i,j,m} F_{k+i-1,l+j-1,m}$$

with $\hat{K}$ being the depthwise convolutional kernel of size $D_K \times D_K \times M$ where the mth filter in $\hat{K}$ is applied to the mth channel in F to produce the mth channel of the filtered output feature map $\hat{G}$. This depthwise convolution has a computational cost of:

$$D_K.D_K.M.D_F.D_F$$

Depthwise separable convolution, which was introduced in [43] and combines depthwise and pointwise convolutions, has, by summing the costs of depthwise and pointwise convolutions, the following computational cost:

$$D_K.D_K.M.D_F.D_F + M.N.D_F.D_F$$

The ratio of the two above equations gives:

$$\frac{D_K.D_K.M.D_F.D_F + M.N.D_F.D_F}{D_K.D_K.M.D_F.D_F} = \frac{1}{N} + \frac{1}{D_k^2}$$

which shows how much the computational cost is reduced.

MobileNetV1 [10] uses 3×3 depthwise separable convolution which requires between 8 to 9 times less computations than standard convolution. The full architecture of MobileNets consists of a regular 3×3 convolution as the very first layer, followed by 13 times the building block in 2.19. There are no pooling layers between these depthwise separable blocks. Instead, some of the depthwise layers have a stride of 2 in order to reduce the spatial dimensions of the data. When that happens, the corresponding pointwise layer also doubles the number of output channels. If the input image is $224 \times 224 \times 3$ then the output of the network is a $7 \times 7 \times 1024$ feature map. The convolution layers are followed by batch normalization. In a classifier based on MobileNets, there is typically a global average pooling layer (see 2.2.3) at the very end, followed by a fully-connected classification layer or an equivalent 1×1 convolution, and a softmax.

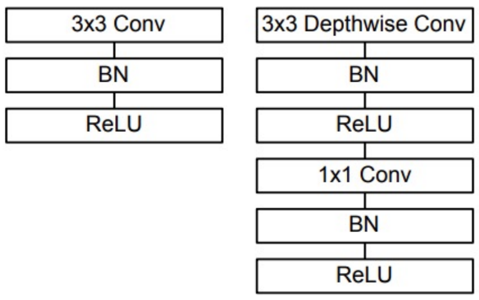


Figure 2.19: Left: Standard convolutional layer with batch normalization and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batch normalization and ReLU. The latter is the building block of MobileNets [10].

As stated at the beginning, MobileNets are a family of NN architectures. Two main hyperparameters define these architectures:

- the depth multiplier α, also known as the "width multiplier". This changes how many channels are in each layer. Using a depth multiplier of 0.5 will halve the number of channels used in each layer, which cuts down the number of computations by a factor of 4 and the number of learnable parameters by a factor 3. It is therefore much faster than the full model but also less accurate.

- the resolution multiplier ρ: we apply this to the input image and the internal representation of every layer is subsequently reduced by the same multiplier. In practice we implicitly set this hyperparameter by setting the input resolution.

MobileNetV2

MobileNetV2 is an improvement of MobileNetV1. It still uses depthwise separable convolutions. However, it also introduces two new features: linear bottlenecks layers and shortcut connections between the bottlenecks. The basic structure of its building block is shown in Figure 2.20. There are 3 convolutional layers in this building block. The last two are a depthwise convolution that filters the inputs, and operates a 1×1 pointwise convolution. However, this time this pointwise convolution makes the number of channels smaller, unlike in MobileNetV1 where the pointwise convolution either kept the number of channels the same or doubled them. This layer is also called a bottleneck layer. The first layer is also a 1×1 convolution. Its purpose is to expand the number of channels in the data before it goes into the depthwise convolution. Therefore this expansion layer has more output channels than input channels: it does the opposite of the bottleneck layer. How much the data gets expanded is given by the hyperparameter expansion factor. Its default value is 6. The input and the output of the block are therefore low-dimensional tensors, while the filtering step that happens inside block is done on a high-dimensional tensor.

The second new feature in MobileNetV2's building block is the residual connection. This works as in ResNet [4] and exists to help with the flow of gradients through the network. Formally, let say we have a neural network block, whose input is x and we want to learn the true distribution $H(x)$. The residual between them is: $R(x) = H(x) - x$. We thus have $H(x) = R(x) + x$. In a residual neural network bloc, there is an identity connection coming from x, and the layers learn the residual $R(x)$ in order to learn $H(x)$.

The activation function used by MobileNetV2 is ReLU6:

$$y = ReLU6(x) = min(max(0, x), 6)$$

This is like the traditional ReLU, but it prevents activations from becoming too big. The reason for this is that ReLU6 is more robust than regular ReLU when using low-precision computation [10]. Moreover, the shape of this function is similar to a sigmoid.

Each layer has batch normalization and the ReLU6 activation. However, the bottleneck layer does not have an activation function. In fact, this layer produces low-dimensional data, and using a non-linearity after this layer destroys useful information according to [11].

The MobileNetV2 architecture is formed of 17 of these building blocks (Figure 2.20) in a row. This is followed by a regular 1×1 convolution, a global average pooling layer (see 2.2.3), and a classification layer. The very first block is slightly different, it uses a regular 3×3 convolution with 32 channels instead of the expansion layer.

2.2.3 Deep transfer learning

Transfer learning [44] allows to build accurate models in a time-saving manner [45]. With transfer learning, instead of starting the training from scratch, the latter starts from patterns that have been learned when solving a different problem in order to leverage previous learning. In computer vision, transfer learning usually consists of using DL models that are pre-trained on a large and diverse benchmark dataset to solve a problem similar to the one that we

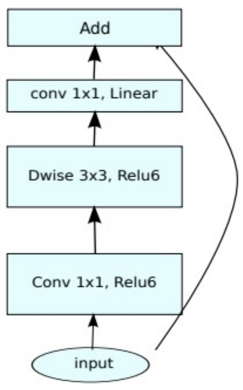


Figure 2.20: The building block of MobileNetV2, which is a bottleneck residual block. Batch normalization is used after every layer [11].

want to solve. The latter is based on the fact that DL models used for image understanding can learn hierarchical feature representations. This means that features learnt by the first layer are general and can be reused in different problems, while features learnt by the last layer are specific and depend on the dataset and task. According to [46]: "if first-layer features are general and last-layer features are specific, then there must be a transition from general to specific somewhere in the network". As a result, the base CNN - especially its lower layers (those that are closer to the inputs) - learn general features, whereas the classifier part, and some of the higher layers of the base CNN, are related to specific features.

When a pre-trained model is used in the context of deep transfer learning, the original classifier is generally removed, then a new classifier that fits the problem is added. Figure 2.21 illustrates this transfer learning. Finally, the model is fine-tuned according to one of three strategies:

- Train the entire model.
- Train some layers and leave the others frozen.
- Freeze the base CNN. The main idea is to keep the base CNN in its original form and then use its outputs to feed the classifier. The pre-trained model is used as a fixed feature extractor. This is particularly useful when computational power for training is limited, the dataset is small, or pre-trained model solves a problem very similar to the one to solve.

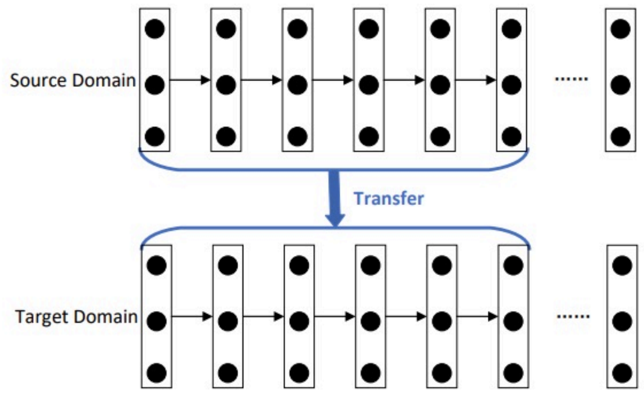


Figure 2.21: Illustration of transfer learning in CNN [47]

Different approaches can be followed to build the classifier that is placed on top of the feature extractor. Some of them are:

- The use of fully-connected layers. For image classification problems, the standard approach is to use a stack of fully-connected layers followed by a softmax activation layer.
- The use of global average pooling. Proposed by [41], in this approach, Instead of adding fully connected layers on top of the feature maps, we take the average of each feature map, and the resulting vector is directly fed into the softmax layer. One advantage of global average pooling over the fully connected layers is that it is more native to the convolution structure by enforcing correspondences between feature maps and categories (concepts). Thus the feature maps can be easily interpreted as categories confidence maps. Another advantage is that there is no parameter to optimize in the global average pooling thus overfitting is avoided at this layer. Furthermore, global average pooling sums out the spatial information, thus it is more robust to spatial translations of the input. We can see global average pooling as a structural regularizer that explicitly enforces feature maps to be confidence maps of concepts (categories).
- The use of linear Support Vector Machines. According to [48], training a linear SVM classifier on top of the extracted features improves classification performance.

Figure 2.22 illustrates the first two approaches.

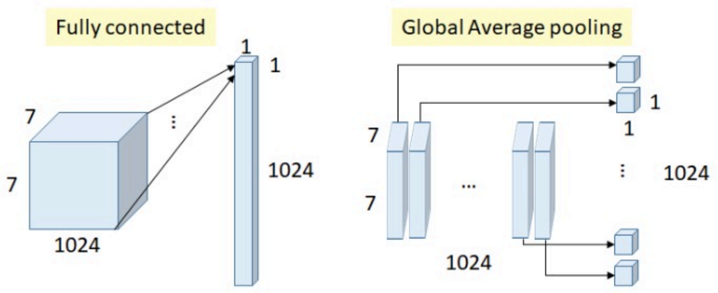


Figure 2.22: Fully connected layers and global average pooling in transfer learning [49]

2.3 Related work

This sections summarize works that are related to this thesis.

2.3.1 Video classification

[23] proposed a two-stream architecture involving a spatial (appearance) stream CNN and temporal stream CNN. The temporal stream took as input stack of horizontal and vertical components of optical flow frames [28] to deal with motion (temporal) information. Optical flow is a local spatio-temporal feature that is useful to extract motion information. Let say that a motion field encodes real world 3D motion, optical flow field is the projection of the motion field onto the 2D image. Therefore, it has two components (vertical and horizontal). The optical flow field consists of a velocity vector for each pixel that shows says how quickly is the pixel moving across the image and in which direction it is moving. An optical flow field is related to two subsequent frames. To obtain it, one should, given two subsequent frames, estimate the apparent motion field between them. This is not simple task, and several methods have been proposed for this [28, 50, 51, 29]. Figure 2.23 illustrates optical flow.

[23] computed optical flow by using the method of [28], which formulated the energy based on constancy assumptions for intensity and its gradient, as well as smoothness of the displacement field. While in [23], the two networks separately capture spatial and temporal information at a fine temporal scale, [24] represents an improvement of this work by investigating on approaches to fuse the two networks over space and time. [23] used CNN-M-2048 [30] on top of single video frames and optical flow images; [24] used VGG-M-2048 and VGG-16 [30, 3] in the two streams.

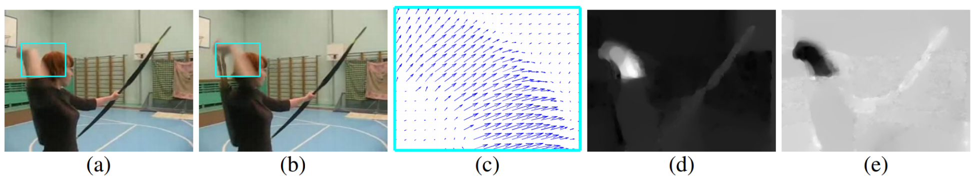


Figure 2.23: Optical flow. (a),(b): a pair of consecutive video frames with the area around a moving hand outlined with a cyan rectangle. (c): a close-up of dense optical flow in the outlined area; (d): horizontal component d^x of the displacement vector field (higher intensity corresponds to positive values, lower intensity to negative values). (e): vertical component d^y . Note how (d) and (e) highlight the moving hand and bow. The input to a CNN contains multiple flows [23].

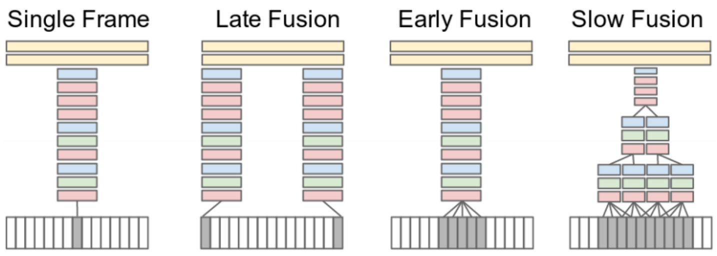


Figure 2.24: Approaches for fusing information over temporal dimension through the network. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. In the Slow Fusion model, the depicted columns share parameters. White/grey rectangles are video frames [32].

[32] used CNNs in order to tackle the video classification problems over the Sports-1M and UCF101 dataset. The study investigated several approaches to fusing information extracted by CNN from video frames across temporal domain. These approaches are: single-frame, Early Fusion, Late Fusion and Slow Fusion, and they are illustrated in 2.24.

[25] used AlexNet[2] and GoogLeNet (Inception-v1) [5] to extract features from individual video frames then proposed to use either some feature pooling methods or LSTM-based methods to combine image information across a video. The study also used optical flow by computing it as in [50] and performed late fusion similar to the two-stream method of [23].

[26] extracted Inception-v3 [27] features from each video frames and proposed to do video classification by using either a Deep Bag of Frame (DBoF)

Pooling based approach or, similar to [25], a LSTM based approach. The study also used another approach that consisted of first computing for each video its video-level features starting from frame-level features of its frames and then using the video-level features to do some machine learning (ML) approaches (Logistic Regression, Hinge Loss and Mixture of Expert).

2.3.2 CNN architectures

[10] introduced the class of efficient models MobileNets, as described in 2.2.2. The study measured performance metrics of these networks with the varying two hyper parameters based on some datasets for some tasks, for example ImageNet for classification or the Stanford Dogs dataset [52] for fine grained recognition. The study even used Faster-RCNN [53] and SSD [54] frameworks with MobileNets to perform object detection.

[11] proposed MobileNetV2, as described in 2.2.2. The study also measured performance metrics on some datasets for some tasks as for MobileNetV1. The study also described the novel framework of object detection, SSDLite. The study finally demonstrated how to build mobile semantic segmentation models through the novel Mobile DeepLabv3 framework.

[5] proposed the GoogLeNet (Inception-v1) architecture. The study set the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14).

[27] proposed the Inception-v2 and Inception-v3 architectures, and benchmarked them on the ILSVRC 2012 classification challenge validation set which allowed to demonstrate substantial gains over the state of the art. The study also reported metrics on the official test set.

2.3.3 Deep transfer learning

[46] experimentally quantified the generality versus specificity of neurons in each layer of a deep convolutional neural network in the context of computer vision. The study found that transferability is negatively affected by two distinct issues:

- the specialization of higher layer neurons to their original task at the expense of performance on the target task, which the authors expected
- optimization difficulties related to splitting networks between co-adapted neurons, which was not expected by the authors.

In an example network trained on ImageNet, the study demonstrated that either of these two issues may dominate, depending on whether features are transferred from the bottom, middle, or top of the network. The study also documented that the transferability of features decreases as the distance between the base task and target task increases, but that transferring features even from distant tasks can be better than using random features. A final surprising result obtained by the paper was that initializing a network with transferred features from almost any number of layers can produce a boost to generalization that lingers even after fine-tuning to the target dataset. Given an conclusion that LeNet, AlexNet, VGG, Inception, ResNet are good chooses in network-based deep transfer learning.

[55] reused front-layers trained by CNN on the ImageNet dataset to compute intermediate image representation for images in other datasets, CNN are trained to learning image representations that can be efficiently transferred to other visual recognition tasks with limited amount of training data.

[56] investigated the transferability of generic representation of an input image at a certain layer of the network trained on a large labeled dataset and the feed-forward units activation, with regard to several factors. It includes parameters for training the network such as its architecture and parameters of feature extraction. The study also showed that different visual recognition tasks can be categorically ordered based on their distance from the source task. The study finally indicated a clear correlation between the performance of tasks and their distance from the source task conditioned on proposed factors.

2.3.4 Quantization of NNs

[17] leveraged low-precision fixed-point arithmetic to accelerate the training speed of CNNs. [18] used 8-bit fixedpoint arithmetic to speed up inference on x86 CPUs. Inspired from these works, [15] proposed a quantization scheme that is used in TensorFlow Lite [22] that focuses on improving the inference speed vs accuracy trade-off on mobile CPUs.

[16] builds on [15] and presented an overview of techniques for quantizing convolutional neural networks for inference with integer weights and activations. The study found that:

- Per-channel quantization of weights and per-layer quantization of activations to 8-bits of precision post-training produces classification accuracies within 2% of floating point networks for a wide variety of CNN architectures.

- Model sizes can be reduced by a factor of 4 by quantizing weights to 8bits, even when 8-bit arithmetic is not supported. This can be achieved with simple, post training quantization of weights.

The study benchmarked latencies of quantized networks on CPUs and DSPs and observed a speedup of $2\times$-$3\times$ for quantized implementations compared to floating point on CPUs. Additionally, it observed speedups of up to $10\times$ on specialized processors with fixed point SIMD capabilities. The study finally found out that:

- quantization-aware training can provide further improvements,reducing the gap to floating point to 1% at 8-bit precision.

- quantization-aware training also allows for reducing the precision of weights to four bits with accuracy losses ranging from 2% to 10%, with higher accuracy drop for smaller networks.

The study introduced tools in TensorFlow and TensorFlowLite for quantizing convolutional networks and reviewed best practices for quantization-aware training to obtain high accuracy with quantized weights and activations. The study finally concluded that it is preferable that per-channel quantization of weights and per-layer quantization of activations be the preferred quantization scheme for hardware acceleration and kernel optimization and proposed that future processors and hardware accelerators for optimized inference support precisions of 4,8 and 16 bits.

Chapter 3

Methodology

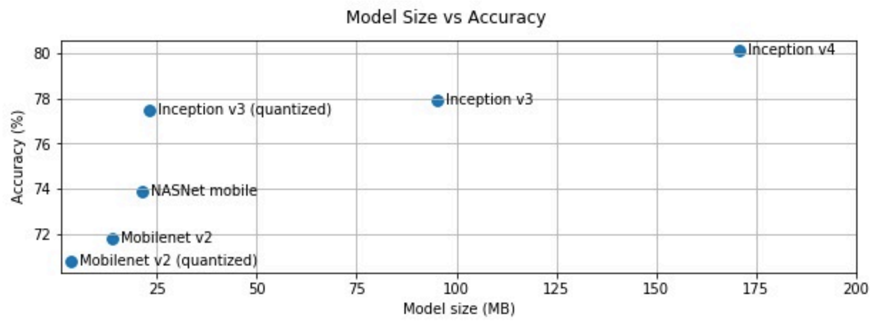
The purpose of this chapter is to provide an overview of the research method that is used in this project. Research method for this degree project is determined following the portal presented in [31].

3.1 Research method

First of all, the positivism philosophical assumption is made, i.e., the reality is objectively given and independent of the observer and instruments. In order to answer the research question, we conduct a qualitative and quantitative research, which involves triangulation. Indeed, to answer the question, video classification pipelines which are indeed new artefacts, are designed and numerically evaluated. They are inspired from existing papers about video classification, but present some differences compared to approaches that are used in them. These differences and the reasons for them are detailed in 4.1. This is a qualitative research because this involves development of new artefacts; this is also a quantitative research because this involves measurement of numerical metrics.

To answer the question, video classification pipelines that are based on quantized MobileNetV2 that is chosen as a representative of memory and computation-efficient CNNs are designed, implemented and evaluated. Indeed, exactly same approach can be recycled when building video classification pipelines that are based on other memory and computation-efficient CNNs, which allows generalization in the conclusions from this thesis. Therefore here, we adopt the case study research strategy. The reasons for the use of quantized MobileNetV2 are that, in a nutshell:

- MobileNetV2 is one of the NNs that are designed for image recognition



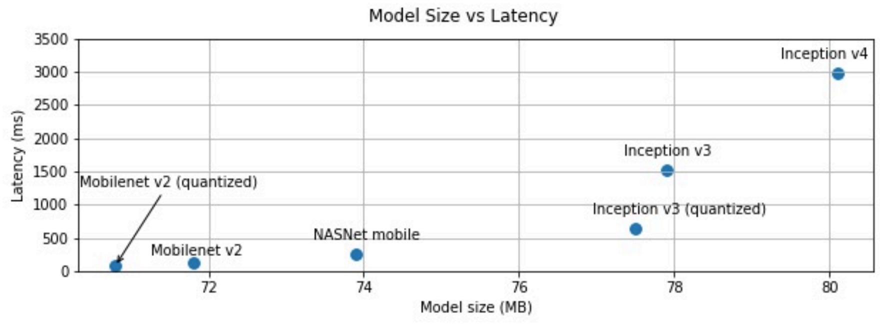


Figure 3.1: Accuracy and latency trade-offs for some popular image classification CNNs [22].

tasks that presents the most effective trade-off between accuracy and model efficiency in terms of model size and computation speed;

- Quantization is one of the most effective techniques that aims to reduce DL model size and computation speed

and they can be combined in order to obtain a highly memory and computation-efficient and accurate enough DL image recognition model. Figure 3.1 and Table 3.1 show this: quantized MobileNetV2 is one of the networks presenting very quick inference time and small model size while being accurate enough.

The pipelines are empirically evaluated based on the empirical research method: appropriate performance metrics are measured by using an appropriate dataset that is collected via experiment and case study data collection method. The multiclass classification metrics explained in 3.3.1 are used. While "performance" is an ambiguous term, because for a classification pipeline this can possibly concern several different aspects such as memory complexity, time complexity end energy consumption, in this study only the performance metrics related to the multiclass classification task are measured. Indeed, it is obvious that the pipelines based on memory and computation-efficient CNN

Model name	Model size	Top-1 accuracy	Top-5 accuracy	TF Lite inference time
Inception_V3	95.3Mb	77.9%	93.8%	1433 ms
Mobilenet_V2_1.0_224_quant	3.4Mb	70.8%	89.9%	80.3 ms

Table 3.1: Performance benchmarks of the pre-trained models optimized to work with TensorFlow Lite. The model files include both TF Lite FlatBuffer and Tensorflow frozen Graph. Performance numbers were benchmarked on Pixel-2 using single thread large core. Accuracy numbers were computed using the TFLite accuracy tool based on ILSVRC 2012 (ImageNet Large Scale Visual Recognition Challenge) image classification task [22, 8].

improve performance in terms of memory size, energy consumption and computational speed. The dataset should serve as a representative of video classification datasets in order to generalize as far as possible obtained results for these datasets and here again, we adopt a case study research strategy. The computational mathematics method is used in order to analyse this dataset. The data analysis, *inter alia*, proves the validity of the used metrics.

To have a reference (baseline) for comparison, similar video classification pipelines but this time based on Inception-v3 are also developed and evaluated based on the same performance metrics and dataset. Indeed, [26] created a novel, large and diverse video dataset based on Inception-v3 features and tested classifiers based on it, and this study is considered as baseline. The comparison between the performance metrics obtained for video classification pipelines based on quantized MobileNetV2 and:

- the metrics obtained from the reference video classification pipelines
- the metrics that are reported in existing papers that used the same dataset

allows to know how well the video classification pipelines based on memory and computation-efficient CNN perform and to validate them. Here too, this is a case study and conclusions obtained for quantized MobileNetV2-based approaches and the dataset are generalized as far as possible to other memory and computation-efficient CNNs and video classification datasets.

Moreover, also single-frame models (similarly to the approach mentioned in 2.3.1) are built on top of both quantized MobileNetV2 and Inception-v3 and evaluated. These models completely ignore temporal information, thus can be used for comparison to verify or falsify the hypothesis that the drop in accuracy induced by the use of computation and memory-efficient CNN at video frames level can be compensated by capturing temporal information via consideration of sequence of these frames. Indeed, based on the metrics (observations) that are measured and collected and the abductive research approach, the first hypothesis that can be verified or falsified is that, the video classification pipelines based on quantized MobileNetV2 perform not as good as the baseline pipelines but they present similar performance and their gap in performance is smaller than the gap in performance between Inception-v3-based and quantized MobileNetV2-based single frame models. If this is the case, it may indicate that there is the aforementioned compensation effect.

The second hypothesis that can be verified or falsified from the collected observations based on the abductive research approach is that, based on the dataset, the video classification pipelines perform not as good as the state-of-the-art results presented in other papers [23, 24, 25] based on very deep and complicated DNNs and/or optical flow, but enough good to be considered acceptable.

There are some methodological limitations. The approaches presented in this thesis do not deal with optical flow, because of its practical limitation mentioned in 1.2. Also, the static frame-level features provide an excellent baseline and constructing compact and efficient motion features is beyond the scope of this thesis that aims to only deal with video frames. Finally, for the same reason, even if some video data can contain sound data, this project does not deal with audio features.

The following sections describe the method application, i.e., how the research method is applied. This includes choice of datasets, softwares and implementations.

3.2 Datasets

The project is based on the UCF101 [57] action recognition dataset that is available online. This is a dataset of 101 action classes from videos uploaded by users on Youtube. They are unconstrained, realistic because contain camera motion, various lighting conditions, partial occlusion and low quality frames, and contain row video data. It contains 13320 clips and 27 hours of video data, each clip belonging to one of the 101 classes. A class also belongs to one of the following types (or categories): Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, Sports [57]. Figure 3.2 visualizes a frame of a clip for each of the classes and indicates which category its class belongs to. Clips of a class are separated into 25 groups, each group containing 4-7 clips that share some common features, for example the background.

Authors of [24] made RGB frames extracted from the UCF101 dataset available on their Github page [58]. This dataset, called the UCF101 RGB frames dataset in all of the following parts, is collected via experiment data collection method and used in this project. This is naturally useful when dealing with frames. This dataset provides input for the video classification pipelines.

Indeed, as it will be detailed in 4.1.1, the artefacts (video classification pipelines) designed in this thesis involve video classification models put on top of freezed CNNs (quantized MobileNetV2 and Inception-v3) that extract features from video frames. Here we exploit transfer learning 2.2.3. Therefore, in order to train only these classification models, frame-level features extracted from every video frames can be stored as .npy (NumPy array) files [59]. By applying different feature extractions based on different CNNs on every frames of the UCF101 RGB frames dataset, we obtain datasets that we call UCF101 frame-level features datasets (they are in plural form because for each of the feature extractors - one based on quantized MobileNetV2 and another one based on Inception-v3 - a frame-level features dataset is collected). These datasets, collected via the experiment data collection method, are input for the classifiers that are put on top of the feature extractors.

The authors of [24] also provided on their Github page [58] the three splits into training and test dataset of the UCF101 dataset that they used for the study described in their paper. The split specified by "ucf101_splits/trainlist01.txt" (train set) and "ucf101_splits/testlist01.txt" (test set) in this repository, referred as “split 1” in [24] and involving 9537 train clips (72% of the total data) and 3783 test clips (28% of the total data), is used in this project. A part in the train set is isolated in order to make it a validation set. Finally, 62% of the total data is used for training, 10% of the total data is used for validation and 28% of the total data is used for testing. The UCF101 RGB frames and UCF101 frame-level features datasets can naturally follow this split.

The UCF101 RGB frames and UCF101 frame-level features datasets are hosted in the cloud computing platform Microsoft Azure [60], leveraging Azure Blob Storage.

Computational mathematics method is used in order to analyse data, in order to determine necessary data pre-processing and prove relevance of the used metrics.

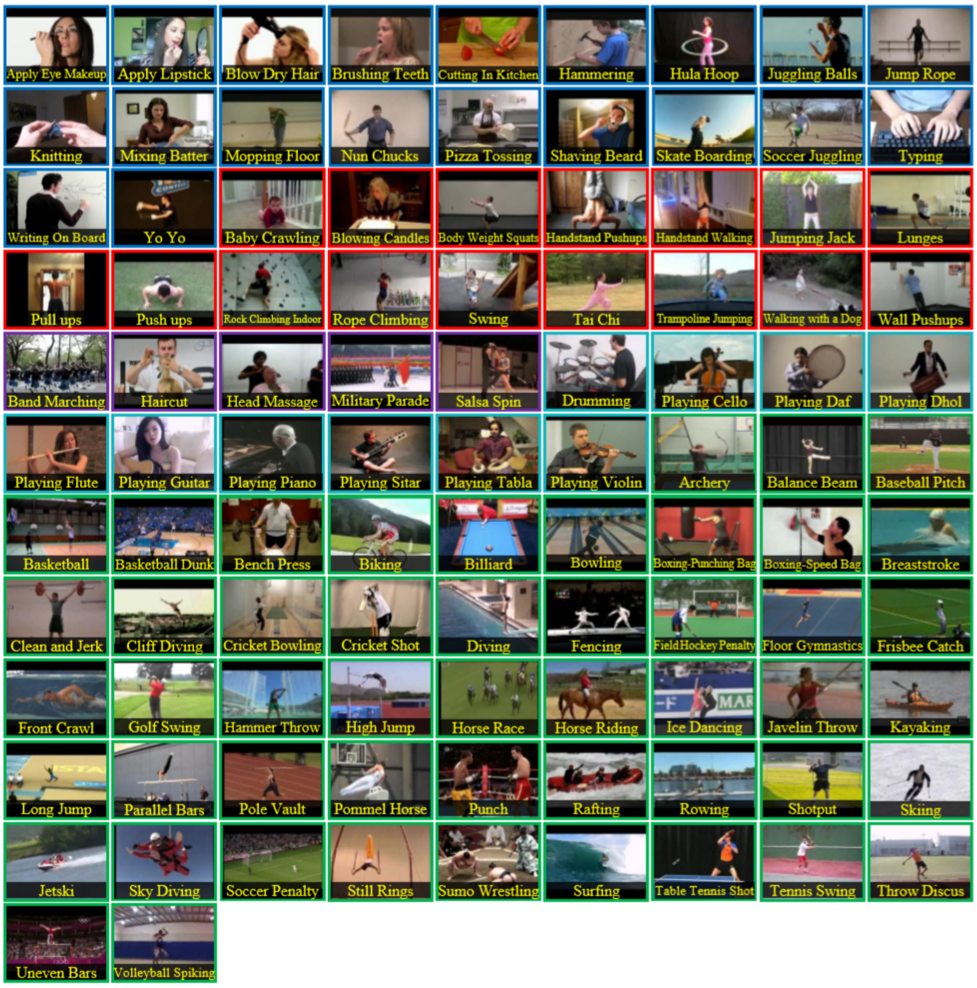


Figure 3.2: 101 classes of the UCF101 dataset. The color of frame borders species to which action type they belong: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, Sports [57].

3.3 Evaluation

This section describes how classification pipelines are evaluated. This allows to know how well the classification pipelines can perform and to validate them.

3.3.1 Evaluation metrics

This section describes the performance metrics that are used in this project in order to assess the classification task done by the pipelines.

Confusion matrix

In a classification problem, there exists a true output y and a model-generated predicted output $\hat{y}$ for each data point. The confusion matrix is $K \times K$, where K is the number of classes. It shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes in the data.

In the context of binary classification problem in which there are 2 possible classes (positive and negative classes), the results for each instance point can be assigned to one of four categories:

- True Positive (TP): the label y is positive and prediction $\hat{y}$ is also positive
- True Negative (TN) : the label y is negative and prediction $\hat{y}$ is also negative.
- False Positive (FP) : the label y is negative but prediction $\hat{y}$ is positive (type I error).
- False Negative (FN) : the label y is positive but prediction $\hat{y}$ is negative (type II error).

and the confusion matrix have the following form: $\begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$

In multiclass problems, a given row of the matrix corresponds to specific value for the "truth". Moreover, one can normalize the confusion matrix by dividing each value by the sum of values in the row the value belongs to. In this way, each value is between 0 and 1. This is interesting in case of class imbalance to have a more visual interpretation of which class is being misclassified.

The following metrics are computed from confusion matrix without normalization.

Accuracy

The accuracy measures how close the prediction is to the true value. We have:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

in case of binary classification problem.

The generalization to multiclass problems is the ratio between number of correctly predicted labels and total number of predictions.

A classifier usually gives a set of predicted labels in a decreasing order of probability and the label with the highest probability is the predicted label (for example a softmax classifier). The ratio between number of cases in which correct labels are in the top k predicted labels and total number of predictions is the top-k accuracy.

However, one should be aware that accuracy is not always a good metric, in case a dataset is highly unbalanced. One can illustrate this based on an example. Assume a highly unbalanced dataset where 95% of the data points are not fraud and 5% of the data points are fraud. A naive classifier that predicts not fraud, regardless of input, will be 95% accurate. For this reason, considering other metrics such as precision, recall and f1-score is also relevant.

Precision

For binary classification,

$$Precision = \frac{TP}{TP + FP}$$

The generalization to multiclass problems is to consider columns of the confusion matrix. Given that a given row of the matrix corresponds to specific value for the "truth", we have:

$$Precision_i = \frac{M_{ii}}{\sum_j M_{ji}}$$

and this is specific for a class i. Precision is the fraction of events where we correctly declared i out of all instances where the algorithm declared i.

Recall

For binary classification,

$$Recall = \frac{TP}{TP + FN}$$

Here again, the generalization to multiclass problems is to consider columns of the confusion matrix. Given that a given row of the matrix corresponds to specific value for the "truth", we have:

$$Recall_i = \frac{M_{ii}}{\sum_j M_{ij}}$$

and this is specific for a class i. Conversely to Precision, recall is the fraction of events where we correctly declared i out of all of the cases where the true of state of the world is i.

f1-score

We have that:

$$f1\text{-}score_i = 2\frac{Precision_i \times Recall_i}{Precision_i + Recall_i}$$

i.e., this metric is the harmonic mean of precision and recall.

3.3.2 Measurement of the evaluation metrics

Indeed, as it will be detailed in 4.1.1, the video classification pipelines involve a feature extractor and a classifier on top of it. The artefacts designed in this thesis involve quantized MobileNetV2 as feature extractor. The accuracy and top-5 accuracy over the test set allow to evaluate the performance of classification task for a combination of a feature extractor and a classifier (after its training) on top of it. Four combinations of a feature extractor and a classifier, as specified in 4.1.1, are subject to this evaluation:

- Quantized MobileNetV2 feature extractor and GRU classifier
- Quantized MobileNetV2 feature extractor and LSTM classifier
- Inception-v3 feature extractor and GRU classifier
- Inception-v3 feature extractor and LSTM classifier

and these metrics are also computed for the single-frame models based on both Quantized MobileNetV2 and Inception-v3 feature extractors. The pipelines that use Inception-v3 as feature extractor are baselines for comparisons.

The measurement of the metrics is done based on Python. Based on obtained values of these metrics, for some of these combinations, confusion matrices based on the test set are plotted and classification reports based on the test set, which shows precision, recall and f1-score for every classes, are produced by using the Scikit-Learn's [61] `classification_report()` function.

3.3.3 Comparison

Metrics are compared between them. The quantized MobileNetV2-based approaches and the reference Inception-v3-based approaches are compared. As aforementioned, the hypothesis that can be verified or falsified based on the abductive research approach is that, the video classification pipelines based on quantized MobileNetV2 not as good as the baseline pipelines in terms of the used metrics, but they present similar performance, and this case may indicate that the drop in accuracy induced by quantized MobileNetV2 can be compensated by capturing temporal information via consideration of sequence of these frames. To support the latter, also a single-frame model is built and evaluated for each of the two feature extraction methods.

The second hypothesis that can be verified or falsified based on the abductive research approach is that, based on the dataset, the video classification pipelines perform not as good as the state-of-the-art results presented in other papers [23, 24, 25] based on very deep and complicated DNNs and/or optical flow, but enough good to be considered acceptable.

3.4 Video classification pipelines implementation

Indeed, as it will be detailed in 4.1.1, the artefacts (video classification pipelines) designed in this thesis involve video classification models put on top of freezed CNNs (quantized MobileNetV2 and Inception-v3) that extract features from video frames.

The designed video classification pipelines are implemented based on Python API of Tensorflow Lite [22] and Keras [62]. TensorFlow Lite allows to use quantized MobileNetV2 whereas Keras allows to use Inception-v3. PIL (Python Imaging Library) [63] and Numpy [59] are used in order to pre-process video clips. Keras is used in order to build and train the the classifiers and the single-frame models on top of feature extractor.

In more details, for the video classification pipelines based on quantized MobileNetV2, the frame-level feature extraction is implemented by using TensorFlow Lite. A subnetwork of quantized MobileNetV2 (with $\alpha = 1.0$), pre-trained in a quantization-aware manner on ImageNet, is used. To do so, the quantized TensorFlow GraphDef of MobileNetV2 with the corresponding two hyperparameters is downloaded from the web page of TensorFlow Lite [22]. The name of the file to download is "Mobilenet_V2_1.0_224_quant". It is "quantized" because it is a float model with FakeQuant ops inserted at the boundaries of fused layers to record min-max range information. This generates a quantized inference workload that reproduces the quantization behavior that was used during its training on ImageNet. A subgraph of this GraphDef is converted into a TensorFlow Lite FlatBuffer for quantized inference. Its input layer is the layer specified the name "input" and its output is the layer specified by the name "MobilenetV2/embedding". As this output is a four dimensional numpy `ndarray`, and it is not a flat feature vector, it then undergoes the global average pooling, implemented based on the numpy function `mean()` [59]. The arithmetic means along the second and third axis are computed. This produces a two dimensional numpy `ndarray` (with its first component being a single-dimensional entry) of data type `float32`. The latter then undergoes a normalization via division by 255. The final `ndarray` obtained in this way, is the feature vector.

For the reference video classification pipelines, a subnetwork of the "floating point" Inception-v3 is used. As it involves floating point inference and not quantized inference, as stated above it requires that input RBG image data be a numpy `ndarray` of data type `float32`, containing float values between -1 and 1 and the output does not need further normalization. A subnetwork of the Keras implementation of Inception-v3 is used. The output already comes from the global average pooling operation so no further processing is done.

To facilitate training of the models in the classification part on top of feature extraction, these steps are applied to every video frames in the UCF101 RGB frames dataset, and produced features are saved as the UCF101 frame-level features datasets, as mentioned in 3.2. This is locally done for the quantized MobileNetV2 feature extraction, based on TensorFlow Lite. More specifically, the TensorFlow devel Docker image tensorflow/tensorflow:nightly-devel is used in order to cross compile and build TensorFlow Lite within a Docker container using this image [22], and the feature extraction is done within this container on a local machine. For Inception-v3 feature extraction, this is done in cloud (Microsoft Azure) by using a Virtual Machine that has a GPU (Standard_NC6 VM) because if done locally the process takes too much time.

The classifiers are trained leveraging Azure Machine Learning Service [64] and Azure Machine Learning SDK for Python, based on the training sets of the UCF101 frame-level features datasets. First, an Azure Machine Learning Workspace is created. Then, an AmlCompute cluster of STANDARD_NC6 GPU VMs is created and attached to the Workspace. Next, we construct an azureml.train.dnn.TensorFlow estimator object, use the GPU cluster as compute target, and pass the mount-point of the the Azure Blob Storage to be used for training and the training code as parameters. The estimator is submitted to the Azure ML experiment to kick off the training. All of the instanced resources on Microsoft Azure are located in the same resource group and the same area (North Europe) in order to avoid data transfer cost and overhead. The host company provided the Visual Studio Enterprise subscription. The trained models are saved and downloaded. This is useful to further plot confusion matrices, compute other metrics over the test set (see 3.3.2) and reconstruct the pipelines. Python is used for this.

3.5 Quality assurance

As the project involves some sources of randomness (weight initialization, possible stochastic gradient descent…), to ensure replicability of this work, random number generators are seeded. The split into training and test sets is detailed and explicitly showed so that the same split can be used. In this chapter and the following chapter, engineering-related contents and all DL-related parameters and hyperparameters are specified and detailed in order to ensure this replicability. Regarding the use of Azure Machine Learning service, the snapshot is also stored and downloaded as part of the experiment in the workspace. In this way, all versions of used libraries are kept. Also, regarding the feature extraction part, all versions of used libraries and docker-related versioning are kept. All of the measurements use reliable and valid deterministic functions.

Chapter 4

Results

This chapter provides, analyses and discusses results that are obtained.

4.1 Video classification pipelines building

In order to answer the research question, the project first aims to build the artefacts: video classification pipelines that are based on quantized MobileNetV2. Also other reference (baseline) pipelines that are based on Inception-v3 are built for comparison purpose. They attribute one of the 101 classes to a clip in the dataset. We have a multiclass classification problem. This section describes how these video classification pipelines are designed and built.

4.1.1 Overview

Figure 4.1 summarizes main components of the video classification pipelines. They involves the following steps:

- input
- frame-level feature extraction
- classification.

The pipelines are composed of a feature extractor and a classifier on top of it, similarly to the approaches used in [25, 26]. Four combinations of a feature extractor and a classifier on top of it are built and used in this project:

- Quantized MobileNetV2 feature extractor and GRU classifier.
- Quantized MobileNetV2 feature extractor and LSTM classifier.

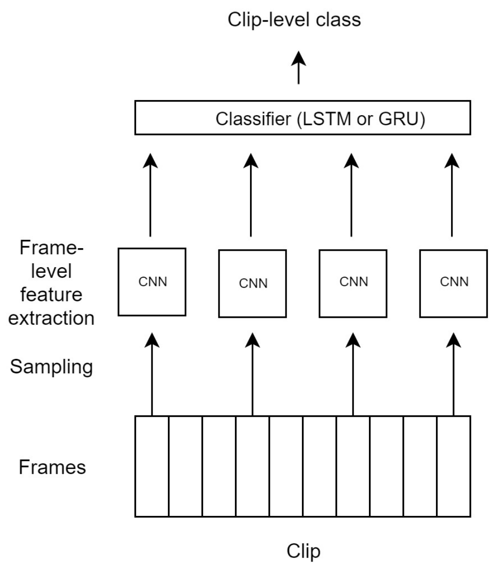


Figure 4.1: Overview of the video classification pipelines

- Inception-v3 feature extractor and GRU classifier.
- Inception-v3 feature extractor and LSTM classifier.

They are inspired from existing papers about video classification:

- [25] used AlexNet[2] and GoogLeNet (Inception-v1) [5] to extract features from individual video frames then proposed to use either some feature pooling methods or LSTM-based methods to combine image information across a video. The study also used optical flow by computing it as in [50] and performed late fusion similar to the two-stream method of [23].
- [26] extracted Inception-v3 [27] features from each video frames and proposed to do video classification by using either a Deep Bag of Frame (DBoF) Pooling based approach or, similarly to [25], a LSTM based approach.

However, the pipelines designed in this thesis present some differences from them:

- we intentionally not deal with optical flow, because the latter presents practical limitation that are mentioned in 1.2 and we believe the static frame-level features provide an excellent baseline. Constructing compact and efficient motion features is beyond the scope of this thesis that aims to only deal with video frames;

- given an input clip, the pipeline does not deal with all of its frames, but instead samples some of them and those sampled frames undergo following processing steps. This way, computational complexity is reduced;
- the pipelines directly benefit from transfer learning (see 2.2.3) by using bottom layers of CNNs that are pre-trained on ImageNet and frozen. This avoids from-scratch training of feature extractors, therefore make training faster and prevents overfitting;
- finally, for our best knowledge, no study about video classification used GRU. Given the fact that GRU presents less parameters than - thus lighter in terms of memory size and faster in terms of computations than - LSTM, we think that GRU deserves to be used in order to do video classification, especially in the context mentioned in Chapter 1 where there are numerous possible applications involving the analysis of video data at the level of devices with limited memory resources and computation capabilities.

We recall that the purpose of this thesis is not to achieve a state-of-the-art result based on the UCF101 dataset.

The following describes each component/step of the designed video classification pipelines.

Input

Input is a set of video frames that are sampled from a video clip which is then classified. $M = 25$ video frames that are uniformly distributed over the clip duration are extracted from the clip. Because a CNN generally only accepts input images having a certain size and having RGB pixel values within a certain interval, sampled video frames are resized and undergo a pixel-normalization scheme in accordance with the later processing steps. For the video classification pipelines based on quantized MobileNetV2, the feature extractor requires that input RBG image data be a numpy `ndarray` of data type `uint8` (i.e., containing integer values from 0 to 255) and shape (224, 224, 3). For the reference pipelines, the Inception-v3 feature extractor requires that input RBG image data be a numpy `ndarray` of data type `float32`, containing float values between -1 and 1 and shape (299, 299, 3). PIL (Python Imaging Library) [63] and Numpy [59] are used in order to satisfy these requirements. Bilinear interpolation is used when resizing the image.

Frame-level feature extraction

Each of the sampled frames in the input is passed to the same CNN that is pre-trained on ImageNet, so that the latter extracts features from them, based on transfer learning (see 2.2.3). These features bring spatial information by individually considering video frames. The memory and computation-efficient CNN is derived from quantized MobileNetV2, that is pre-trained in a quantization-aware manner on ImageNet [9]. The latter has the same architecture as MobileNetV2 with $\alpha = 1$ (see 2.2.2), except that:

- at each layer a fake quantization node is added, in order to allow quantization-aware training as described in 2.1.6 .
- the final classification layer is deleted thus the output comes from the global average pooling layer.

The CNN used in the reference video classification pipelines is derived from Inception-v3 that is pre-trained on ImageNet, as [26] extracted Inception-v3 features from video frames to make a dataset of videos for classification and this study is considered as baseline. An architecture similar to the one of Table 2.1 is used, except that the last two layers are deleted and the pooling layer - which is in reality a global average pooling layer - gives the output.

Quantized MobileNetV2 requires an input RGB image size of 224×224 whereas Inception-v3 requires an input RGB image size of 299×299. Therefore video frames are resized beforehand in accordance with this. Also, RGB pixel values are normalized in accordance with what their implementations expect (see 3.4).

Classification

The frame-level features are aggregated in order to capture temporal information and produce class prediction. In this project, for each of the two feature extraction methods, two classification methods based on RNNs are tested: the first one is based on LSTM, and the another one is based on GRU. The choice of using LSTM comes from the fact that in [26] the best performing approach was based on LSTM according to the metrics they used. LSTM, by operating on frame-level CNN features, can learn how to integrate information over time thus is capable of learning from temporally ordered sequences by explicitly considering sequences of CNN features. Since videos contain dynamic content, the variations between frames may encode additional information which could be useful in making more accurate predictions. GRU is based on the same idea, but it presents less parameters than LSTM thus is less prone to overfitting and is more suitable to deploy on a device with limited computation and memory capabilities. For our best knowledge, no studies about video classification used GRU to tackle video classification.

The two methods consist of putting either a LSTM or GRU layer as the first layer on top of the feature extraction, and then putting additional layers to do the classification. For both approaches:

- The second layer is a batch normalization layer.
- The third layer is a hidden dense layer with 512 neurons. At this level, the activation is linear.
- The fourth layer is again a batch normalization layer.
- The fifth layer is the LeakyReLu activation function, with $\alpha = 0.3$.
- The last layer is a dense layers with 101 neurons that does softmax classification.

Between the last and the fifth layers, dropout with probability = 0.5 is applied at training time. Batch normalizations are added before the activation function of the previous layer, in accordance with the original paper that introduced the method [37].

Regarding the first layer, for both approaches, the dimensionality of input is D and the length of input sequences is $M = 25$. $D = 1280$ when quantized MobileNetV2 is used as feature extractor; $D = 2048$ when Inception-v3 is used as feature extractor. The dimensionality of the output space is also D. At training time, 0.5% of the units are dropped out for the linear transformation of the inputs. Only the last output in the output sequence is returned.

All other parameters for all layers are the default parameters that are used in Keras. These architectures were empirically found to achieve enough good accuracy while involving relatively small number of parameters and inference time.

Besides, in order to help checking the hypothesis that the drop in accuracy induced by the use of quantized MobileNetV2 when dealing with individual frames can be compensated by capturing temporal information via consideration of sequence of these frames, also a single-frame model is tested for each of the two feature extraction methods. Indeed, single-frame model only captures spatial frame-level information. If the gaps between performance of classifications based on the two feature extraction methods are lower for the RNN-based classification approaches than the single-frame models, this can support the validation of this hypothesis. The single-frame model is a FNN that takes as input a frame-level feature from a clip. It has the following architecture:

- The first layer is a hidden dense layer with 512 neurons. At this level, the activation is linear.
- The second layer is a batch normalization layer.
- The third layer is the LeakyReLu activation function, with $\alpha = 0.3$.
- The last layer is a dense layers with 101 neurons that does softmax classification.

Here too, between the last layer and the third layer, dropout with probability = 0.5 is applied at training time. The other parameters are default parameters that are used in Keras.

4.1.2 Model training

LSTM-based/GRU-based/single-frame models in the classification part are trained, based on the training sets of the UCF101 frame-level features datasets. For all models, categorical cross-entropy is chosen as loss function. Adam optimizer with learning rate = 0.00001 and decay=0.000001 is used. Right numbers of epochs are determined based on early stopping (see 2.1.5), by using the validation set. The trained models are saved and downloaded. This is useful to further compute metrics over the test set and plot confusion matrices.

4.2 Data Analysis

This section provides some results of the computational mathematics method-based analysis of the UCF101 dataset. This analysis can be naturally extended for the UCF101 RGB frames and UCF101 frame-level features datasets. Table 4.1 summarizes characteristics and statistics of the dataset, such as minimum/maximum clip length, frame rate and resolution. Figure 4.2 shows number of clips per class and the distribution of clip durations.

One can deduce from Table 4.1 that the minimum number of frames per clip is $\lfloor 25 \times 1.06 \rfloor = 26$. Therefore the choice of $M = 25$ in the previous chapter is judicious. Regarding resolution, as the (quantized) MobileNetV2 architecture used in this project requires an input resolution of 224×224, there is downsampling for the memory and computation-efficient feature extraction. The Inception-v3 feature extraction requires an input resolution of 299×299, so it involves horizontal downsampling and vertical upsampling (via bilinear interpolation, as specified in 3.4).

Regarding Figure 4.2, in our project, only the number of clips per class is relevant, as same number of frames is sampled from a clip to give it a class. The distribution of numbers of clips per class gives hints about relevance of performance metrics. From simple computations, one can infer that the maximum and minimum numbers of clips per class are respectively 160 and 95 and roughly speaking, number of clips per class is evenly distributed over the classes according to Table 4.1, which means all of the metrics that are used in this project can be considered as relevant.

Actions	101
Clips	13320
Groups per Action	25
Clips per Group	4-7
Mean Clip Length	7.21 sec
Total Duration	1600 mins
Min Clip Length	1.06 sec
Max Clip Length	71.04 sec
Frame Rate	25 fps
Resolution	320 X 240
Audio	Yes (51 actions)

Table 4.1: summary of the UCF101 dataset [57].

4.3 Classification results

This section describes the observations, i.e, the performance metrics related to the classification task that are measured from the video classification pipelines.

Table 4.3 shows accuracy and top-5 accuracy metrics that are obtained for the four combination of feature extractor and classifier specified in 3.3.2. According to it, GRU performs better than LSTM. GRU has less parameters than

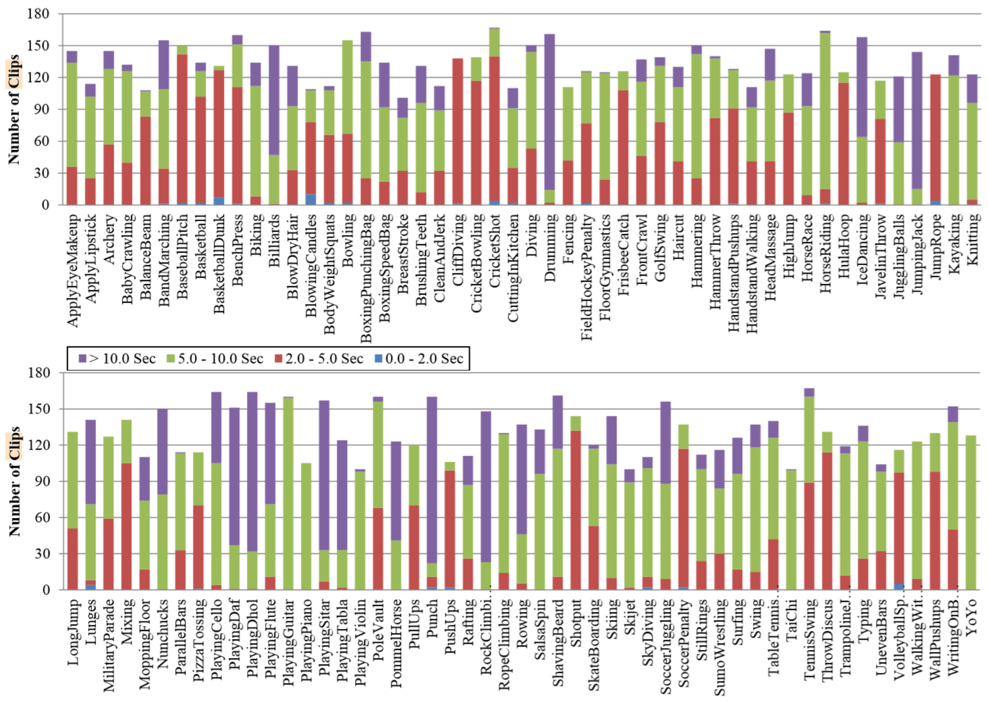


Figure 4.2: Number of clips per class. The distribution of durations is illustrated by the colors [57].

Feature extractor	Classifier	Accuracy	top-5 accuracy
Inception-v3	LSTM	0.7875	0.9422
Inception-v3	GRU	0.7961	0.9469
Quantized MobileNetV2	LSTM	0.7430	0.9117
Quantized MobileNetV2	GRU	0.7594	0.9297
Inception-v3	Single Frame	0.6969	0.8812
Quantized MobileNetV2	Single Frame	0.6425	0.8473

Table 4.2: Classification results

LSTM thus is less prone to overfitting. Therefore in the context of deployment of video classification pipeline on a computationally limited platform, GRU should be preferred to LSTM and this result is in accordance with this.

For the above reason, the combinations of GRU and the two feature extraction methods are used in order to plot confusion matrices and compute additional performance metrics (precision, recall and f1-score) over the test set:

- Normalized confusion matrices for the combinations of GRU and each of the two feature extraction methods are plotted (Figure 4.3 and Figure 4.4). Numbers of correct and incorrect predictions are also shown in the confusion matrices without normalization in Appendix A. Figure 4.5 and Figure 4.6 are parts of these normalized confusion matrices restricted to the 3 classes: 'ApplyEyeMakeup', ApplyLipstick' and 'Archery'.

- We also partly show in the following their classification reports, reporting precision, recall and f1-score for each of the 101 classes and their average values over the 101 classes (page 63). Their entire contents are available in Appendix A.

Feature extractor	Classifier	Precision	Recall	f1-score
Inception-v3	Single-frame	0.67	0.67	0.66
Quantized MobileNetV2	Single-frame	0.62	0.61	0.61
Inception-v3	GRU	0.77	0.77	0.76
Quantized MobileNetV2	GRU	0.74	0.73	0.72

Table 4.3: Average precision, recall and f1-score over the 101 classes

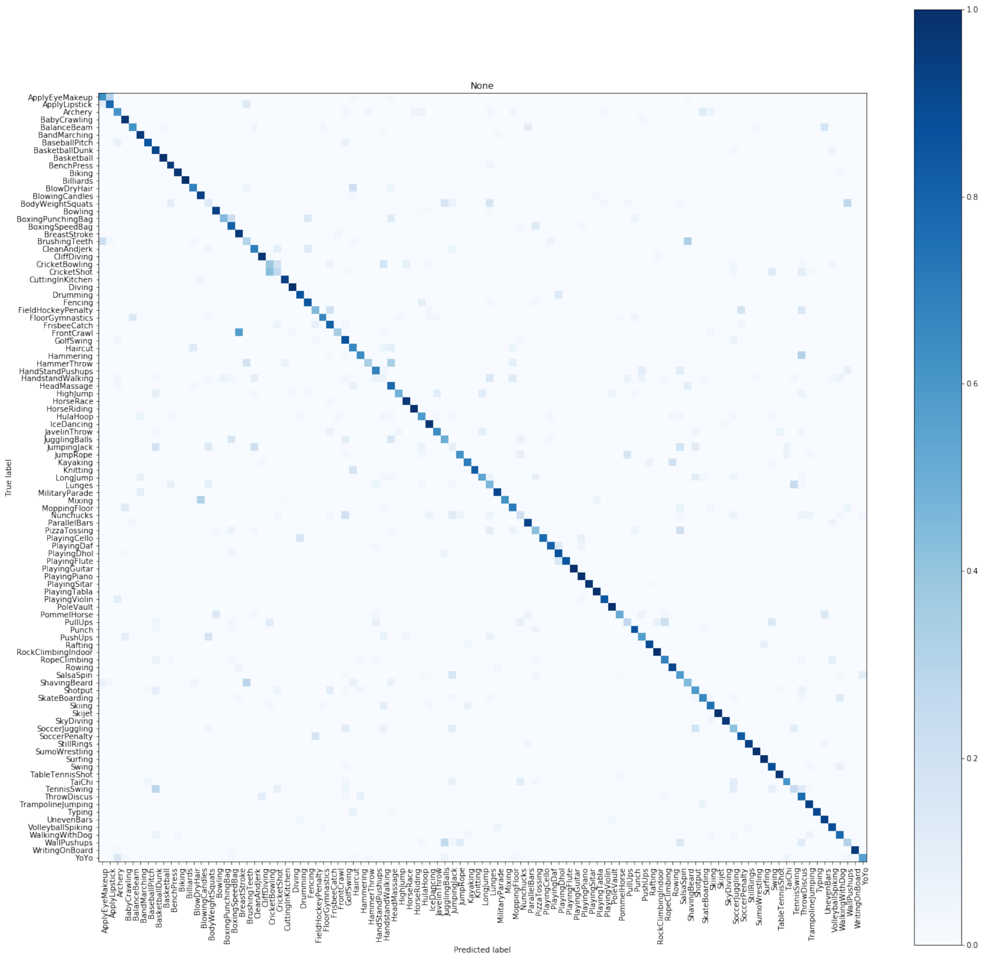


Figure 4.3: Normalized confusion matrix (quantized MobileNetV2 feature extraction and GRU classifier). Color depth indicates proportions of predictions.

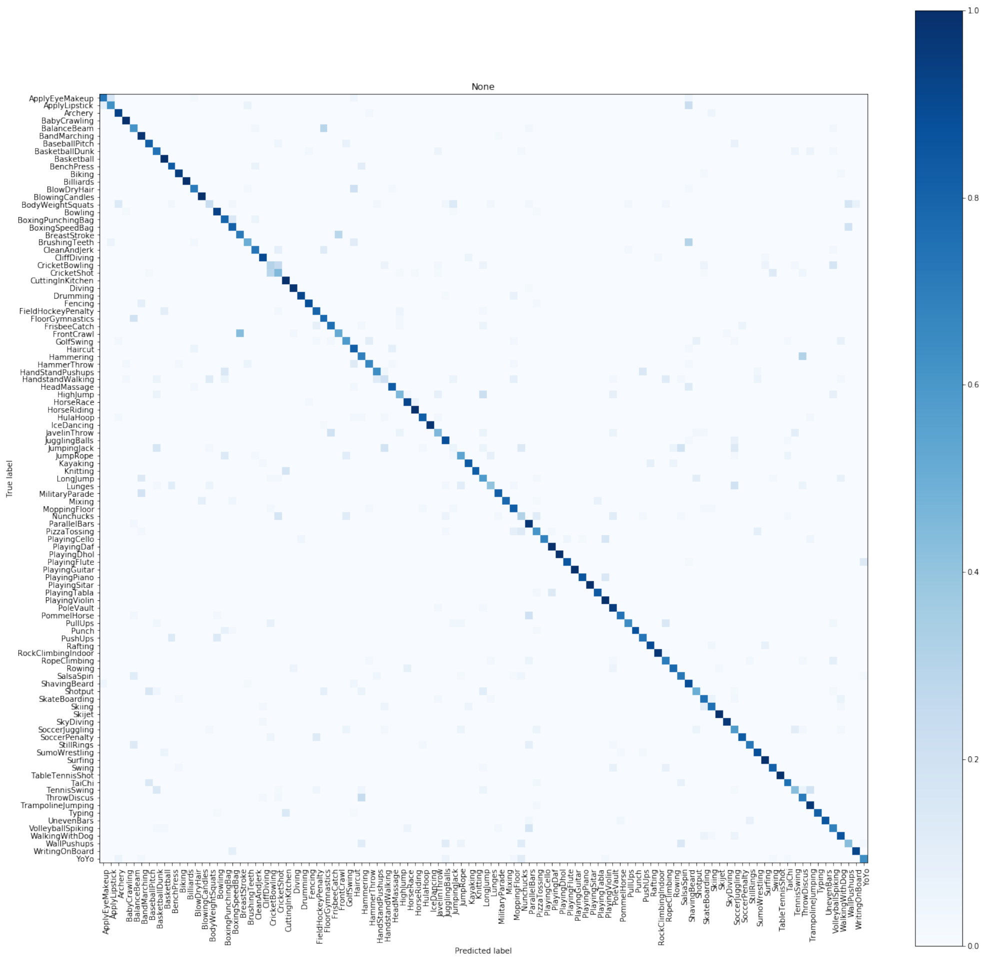


Figure 4.4: Normalized confusion matrix (Inception-v3 feature extraction and GRU classifier). Color depth indicates proportions of predictions.

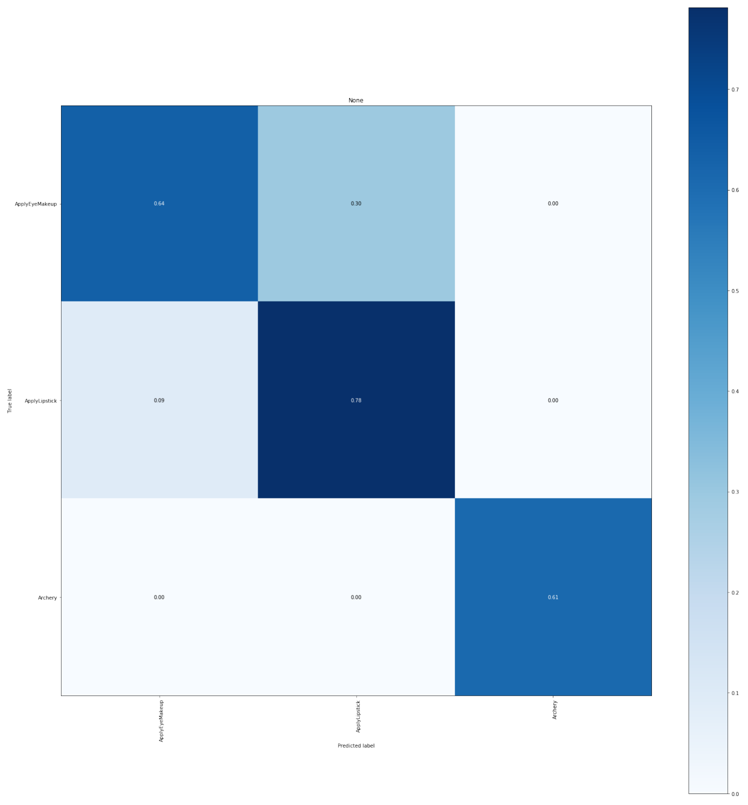


Figure 4.5: Normalized confusion matrix (quantized MobileNetV2 feature extraction and GRU classifier) for the 3 classes: 'ApplyEyeMakeup', 'ApplyLipstick' and 'Archery', along with proportions of correct and incorrect predictions. Color depth indicates proportions of predictions.

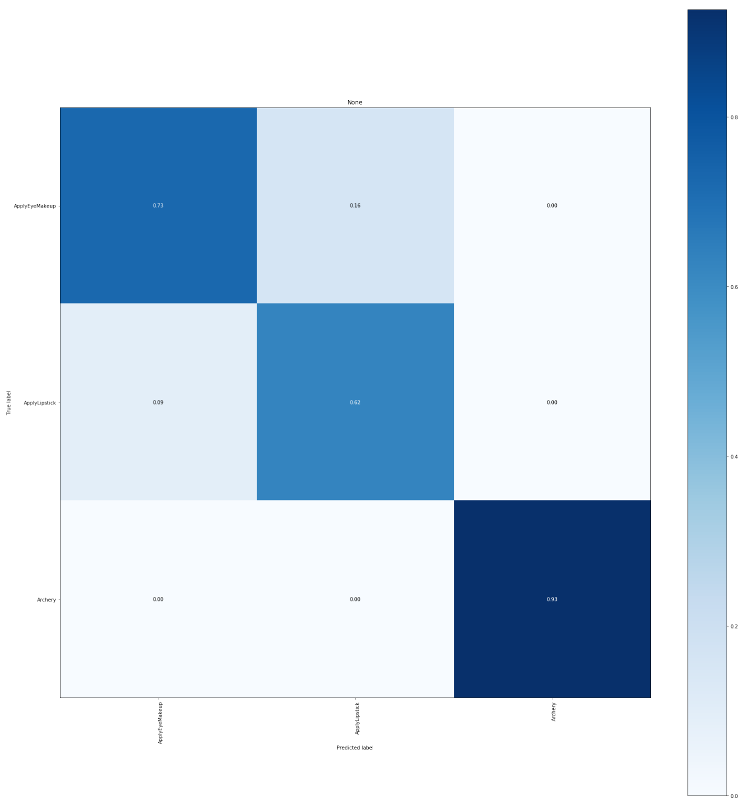


Figure 4.6: Normalized confusion matrix (Inception-v3 feature extraction and GRU classifier) for the 3 classes: 'ApplyEyeMakeup', 'ApplyLipstick' and 'Archery', along with proportions of correct and incorrect predictions. Color depth indicates proportions of predictions.

Partial classification report (quantized MobileNetV2 feature extraction and GRU classifier)

```
                precision    recall  f1-score   support

ApplyEyeMakeup       0.67      0.64      0.65        44
 ApplyLipstick       0.62      0.78      0.69        32
       Archery       0.62      0.61      0.62        41
  BabyCrawling       0.79      0.97      0.87        35
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
   avg / total       0.74      0.73      0.72      3783
```

Partial classification report (Inception-v3 feature extraction and GRU classifier)

```
                precision    recall  f1-score   support

ApplyEyeMakeup       0.86      0.73      0.79        44
 ApplyLipstick       0.69      0.62      0.66        32
       Archery       0.83      0.93      0.87        41
  BabyCrawling       0.92      1.00      0.96        35
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
         .             .         .         .         .
   avg / total       0.77      0.77      0.76      3783
```

4.4 Result analysis and discussion

Regarding the UCF101 dataset,

- The LSTM-based approach combined with the use of optical flow in [25] achieved a maximum 3-fold accuracy of 88.6%.
- [23] achieved the 3-fold accuracy of 87.0% with its optical flow-based two stream approach.
- [24] achieved a maximum 3-fold accuracy of 92.5% with its optical flow-based two stream approach combined with its fusion method.

and accuracy values obtained in this project seem to demonstrate globally lower performance (even if we only use one of the 3 splits that were used in these studies). However, the above studies used optical flow, which can be considered as so far the best descriptor giving motion information. They also used some data augmentation strategies. Moreover, [25] used video classification models that were pre-trained on the larger Sports-1M dataset [32], and deeper and more complicated NN with five layers of LSTM, which should have resulted in massive number of parameters. Last but not least, all of these studies fine-tuned the feature extraction parts based on the augmented UCF101 dataset whereas in this thesis the feature extractors were pre-trained on ImageNet and were not fine-tuned. What is worth mentioning is that our approaches based on quantized MobileNetV2 features gave satisfactory performance metrics that do not differ so much from our reference approaches that are based on Inception-v3 features (Table 4.3 and the classification report in 4.3):

- For the GRU-based approaches, switching from Inception-v3 features to quantized MobileNetV2 features caused 3.67% of decrease in the accuracy and 1.72% of decrease in the top-5 accuracy. For LSTM-based approaches, switching from Inception-v3 features to quantized MobileNetV2 features caused 4.45% of decrease in the accuracy and 3.05% of decrease in the top-5 accuracy. For single-frame models, switching from Inception-v3 features to quantized MobileNetV2 features caused 5.44% of decrease in the accuracy and 3.39% of decrease in the top-5 accuracy. Regarding the image classification based on ImageNet, the difference in accuracy and top 5 accuracy between the Inception-v3 and quantized MobileNetV2 are respectively about 8% and 4% according to Table 3.1.

- For the three other metrics - precision, recall and f1-score - and the GRU-based approaches, according to the classification reports in 4.3 and Table 4.3, their average values decrease of 3% for precision and 4% for the other metrics when switching from quantized MobileNetV2 feature extractor to Inception-v3 feature extractor. For single-frame models, their average values decrease of 6% for recall and 5% for the other metrics when switching from quantized MobileNetV2 feature extractor to Inception-v3 feature extractor.

The gaps between the metrics obtained via the two feature extraction methods (quantized MobileNetV2 and Inception-v3) are lower for RNN-based approaches than single-frame models. This shows to some extent that the drop in accuracy induced by the use of quantized MobileNetV2 when dealing with individual frames can be compensated by capturing temporal information via consideration of sequence of these frames. However, this compensation is not enough so that video classifications based on the two feature extractors perform same. Inception-v3 features-based approach still globally present better performance. On the one hand, this should stem from intrinsic gap in accuracy between these networks. On the other hand, for the UCF101 dataset, spatial information already discriminate well different classes, because the dataset contains high quality data: short, well-segmented videos of concepts that can typically be identified in a single frame. This is evidenced by the already high performance of single-frame models (Table 4.3). The same remark has been stated in [25].

Besides, more fine-grained analysis of the classification reports (in Appendix A) gives interesting results. Among the evaluated 303 metrics (precision, recall and f1-score for all of the 101 classes), 157 metrics are higher when Inception-v3 is used as feature extractor whereas 109 metrics are higher when quantized MobileNetV2 is used as feature extractor, and 37 metrics are equal. Among the 101 classes, there are 54 classes for which at least one of the three metrics is higher when quantized MobileNetV2 is used as feature extractor. Among them, there are 23 classes for which all of the three metrics is either equal or higher when quantized MobileNetV2 is used as feature extractor than when Inception-v3 is used as feature extractor. Again among them, there are 15 classes for which all of the three metrics is higher when quantized MobileNetV2 is used as feature extractor than when Inception-v3 is used as feature extractor and 7 classes for which all of the three metrics are equal for both feature extractors. These facts show that regarding the three metrics, the quantized MobileNetV2 feature-based approach can perform better than the Inception-v3 feature-based approach for some classes. Figure 4.5 and Figure 4.6 illustrate the latter: the Inception-v3 feature-based approach recognizes better the actions ’ApplyEyeMakeup’ and ’Archery’ - which was expected-, whereas the quantized MobileNetV2-based approach recognizes better the action ’ApplyLipstick’.

Chapter 5

Conclusion and future work

To answer the research question, we try to generalize as far as possible our case study. To answer the first part of the research question, video classification pipelines based on memory and computation-efficient CNN - quantized MobileNetV2 being used in our case study- can be built by adopting the approach that is developed in this thesis: by adopting an approach similar to the ones in [25, 26], sampling video frames from video, extracting features from them based on transfer learning and aggregating them while capturing temporal information by using either GRU or LSTM that are trained. In our case study, quantized MobileNetV2 is chosen as a representative of memory and computation-efficient CNNs, but similar approaches can be recycled when building video classification pipelines that are based on other memory and computation-efficient CNNs, such as MobileNetV1, MobileNetV2, quantized MobileNetV1 and others.

To answer the second part of the research question, our approaches based on quantized MobileNetV2 globally achieve satisfactory performance according the metrics that are used (accuracy, top-5 accuracy, precision, recall and f1-score), which validate the artefacts. Based on the UCF101 dataset, that is used as a representative video classification datasets, our pipelines based on quantized MobileNetV2 achieved satisfactory performance metrics that do not differ so much from our reference pipelines that are based on Inception-v3. The gaps between metrics obtained via quantized MobileNetV2 and Inception-v3 feature extraction methods are lower for RNN-based approaches than single-frame models, which may, to some extent, validate the hypothesis that the drop in accuracy induced by the use of quantized MobileNetV2 when dealing with individual frames can be compensated by capturing temporal information via consideration of sequence of these frames. Regarding these results, video classification pipelines that are based on other memory and computation-efficient networks than quantized MobileNetV2 but that follow the same architecture and transfer learning approach should record similar performance metrics and trends and bring similar conclusions. However, the aforementioned compensation is not enough so that the video classifications based on a memory and computation-efficient CNN and larger and more accurate CNN perform same. On the one hand, this should stem from intrinsic gap in accuracy between these networks. On the other hand, for the UCF101 dataset, spatial information already discriminate well different classes, because the dataset contains high quality data: short, well-segmented videos of concepts that can typically be identified in a single frame. This is evidenced by the already high performance of single-frame models. The same remark has been stated in [25]. Therefore video classification based on an accurate image recognition CNN should record better performance when using the UCF101 dataset. To some extent, the case study should generalise to other video classification datasets, show similar trends and bring similar conclusions. Especially, the aforementioned gap should be reduced when images in these datasets that bring spatial information present less diversity. The use of memory and computation-efficient CNNs may be very effective in this case.

As pointed out in the last part of the previous chapter, there are non-negligible numbers of classes for which average precision, recall or f1-score are better when quantized MobileNetV2 is used as feature extractor than when Inception-v3 is used as feature extractor. This fact pointed out in this case study should be also observable when a memory and computation-efficient CNN other than quantized MobileNetV2 is used as feature extractor. As future work, a deeper investigation on the possible reasons for this phenomenon should be interesting.

The use of the artefacts that are developed in this thesis should obviously be relevant when dealing with multi-class classification of videos. Therefore the conclusions can carry over other applications of video classification than action recognition, for example dynamic scene recognition. Besides, computation and memory-efficient CNN can be used for some tasks that deal with video data but that are different from video classification, for example video (temporal) segmentation. This is the process of partitioning a video sequence into disjoint sets of consecutive frames that are homogeneous according to some defined criteria. Features extracted from computation and memory-efficient CNN can be used in order to tackle this task, and this can be useful for one who wants to perform it on a computationally limited platform.

By answering the research question, this thesis shows, via performance metrics measured over the UCF101 dataset, the potential utility of memory and computation-efficient CNN when dealing with video classification, which is interesting for one who wants to do video classification on a platform that is limited in terms of memory and computation capabilities. This work can serve as a significant starting point for video classification pipelines that are based on memory and computation-efficient CNNs.

In this thesis work, pre-trained feature extractor is not fine-tuned. In the future, one should try to fine-tune it in order to further increase performance of video classification. Quantized feature extractor should be fine-tuned in a quantization-aware manner. Besides, this thesis work does not use data augmentation strategies. Some of them can be used in order to further obtain better results.

The combination of quantized MobileNetV2 and a RNN - preferably GRU - forms the video classification pipeline based on computation and memory-efficient CNN in this thesis. However, this RNN classifier can be further compressed, based on approaches such as knowledge distillation [20]. This is suitable for more effective deployment of video classification pipeline on devices with limited memory and computational power, because both GRU and LSTM still involves large number of parameters (even if GRU has less parameters than LSTM). Currently, GRU and LSTM are not supported by TensorFlow Lite for quantization. In the future, when they will be, the video classification pipeline should be updated by doing quantization-aware training of its classifier, in order to further compress it.

Also, the video classification pipeline can be improved by doing a knowledge transfer from another video classification model that is trained on larger and more diverse dataset. If one wants to leverage quantized MobileNetV2 features, Youtube-8m [26] may not be a good candidate as it is a dataset of Inception-v3 features. Recently, IBM released a large-scale human-annotated collection of one million short videos corresponding to dynamic events unfolding within three seconds, the Moments in Time Dataset [65]. This dataset may be a good starting point for this knowledge transfer. Indeed, one of the major reasons for the delay in research improvement between image understanding and video understanding is that video understanding lacks enough diverse and large datasets that can be equivalent to for example ImageNet in image understanding. In correlated way, unlike in image understanding, there is lack of popular networks and benchmarking studies in video understanding. This kind of datasets should be more developed and additional benchmarking studies should be done. The Moments in Time Dataset seems to be a good starting point for this.

Finally, in this thesis, unlike in [26], video representation-based approaches using quantized MobileNetV2 features were not tested. They can be interesting approaches to try in the future. However, in the context of multiclass classification problem, they may not be memory and computation-efficient enough to be deployed on a platform with limited memory and computational capabilities, as they first require defining rigorous framework of multiclass classification (for example, one versus one or one versus all) that is less flexible than a softmax classifier.

Bibliography

[1] Yann LeCun, Y Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[3] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[4] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[5] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[6] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *CoRR* abs/1602.07261 (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261.

[7] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4. URL: https://doi.org/10.1007/s11263-009-0275-4.

[8] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[9] J. Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[10] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[11] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[12] Ningning Ma et al. “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design”. In: *CoRR* abs/1807.11164 (2018). arXiv: 1807.11164. URL: http://arxiv.org/abs/1807.11164.

[13] Forrest N. Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size”. In: *CoRR* abs/1602.07360 (2016). arXiv: 1602.07360. URL: http://arxiv.org/abs/1602.07360.

[14] Xiangyu Zhang et al. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: *CoRR* abs/1707.01083 (2017). arXiv: 1707.01083. URL: http://arxiv.org/abs/1707.01083.

[15] Benoit Jacob et al. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”. In: *CoRR* abs/1712.05877 (2017). arXiv: 1712.05877. URL: http://arxiv.org/abs/1712.05877.

[16] Raghuraman Krishnamoorthi. “Quantizing deep convolutional networks for efficient inference: A whitepaper”. In: *CoRR* abs/1806.08342 (2018). arXiv: 1806.08342. URL: http://arxiv.org/abs/1806.08342.

[17] Suyog Gupta et al. “Deep Learning with Limited Numerical Precision”. In: *CoRR* abs/1502.02551 (2015). arXiv: 1502.02551. URL: http://arxiv.org/abs/1502.02551.

[18] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. “Improving the speed of neural networks on CPUs”. In: *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*. 2011.

[19] Yu Cheng et al. “A Survey of Model Compression and Acceleration for Deep Neural Networks”. In: *CoRR* abs/1710.09282 (2017). arXiv: 1710.09282. URL: http://arxiv.org/abs/1710.09282.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. “Distilling the Knowledge in a Neural Network”. In: *NIPS Deep Learning and Representation Learning Workshop*. 2015. URL: http://arxiv.org/abs/1503.02531.

[21] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. “Model Compression”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’06. Philadelphia, PA, USA: ACM, 2006, pp. 535–541. ISBN: 1-59593-339-5. DOI: 10.1145/1150402.1150464. URL: http://doi.acm.org/10.1145/1150402.1150464.

[22] Google. *TensorFlow Lite*. URL: https://www.tensorflow.org/lite.

[23] Karen Simonyan and Andrew Zisserman. “Two-Stream Convolutional Networks for Action Recognition in Videos”. In: *CoRR* abs/1406.2199 (2014). arXiv: 1406.2199. URL: http://arxiv.org/abs/1406.2199.

[24] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Convolutional Two-Stream Network Fusion for Video Action Recognition”. In: *CoRR* abs/1604.06573 (2016). arXiv: 1604.06573. URL: http://arxiv.org/abs/1604.06573.

[25] Joe Yue-Hei Ng et al. “Beyond Short Snippets: Deep Networks for Video Classification”. In: *CoRR* abs/1503.08909 (2015). arXiv: 1503.08909. URL: http://arxiv.org/abs/1503.08909.

[26] Sami Abu-El-Haija et al. “YouTube-8M: A Large-Scale Video Classification Benchmark”. In: *CoRR* abs/1609.08675 (2016). arXiv: 1609.08675. URL: http://arxiv.org/abs/1609.08675.

[27] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[28] T. Brox et al. “High accuracy optical flow estimation based on a theory for warping”. In: *European Conference on Computer Vision (ECCV)*. Vol. 3024. Lecture Notes in Computer Science. Springer, May 2004, pp. 25–36. URL: http://lmb.informatik.uni-freiburg.de/Publications/2004/Bro04a.

[29] Norazlin Ibrahim et al. “Implementation of Differential Optical Flow Algorithms in Natural Rigid Video Motion”. In: *Lecture Notes in Engineering and Computer Science* 2174 (Mar. 2009).

[30] Ken Chatfield et al. “Return of the Devil in the Details: Delving Deep into Convolutional Nets”. In: *CoRR* abs/1405.3531 (2014). arXiv: 1405.3531. URL: http://arxiv.org/abs/1405.3531.

[31] Anne Håkansson. “Portal of Research Methods and Methodologies for Research Projects and Degree Projects”. In: 2013.

[32] A. Karpathy et al. “Large-Scale Video Classification with Convolutional Neural Networks”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. June 2014, pp. 1725–1732. DOI: 10.1109/CVPR.2014.223.

[33] Amir Payberah. *Large Scale Machine Learning and Deep Learning*. KTH Royal Institute of Technology. Nov. 1, 2018. URL: https://id2223kth.github.io/ (visited on 11/01/2018).

[34] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[35] Junyoung Chung et al. “Gated Feedback Recurrent Neural Networks”. In: *CoRR* abs/1502.02367 (2015). arXiv: 1502.02367. URL: http://arxiv.org/abs/1502.02367.

[36] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/?fbclid=IwAR04RSqY6CSlqMq1yMAUG630njDpAJqznxIHLOsIsDTipE5VG7bTJPaS6As (visited on 05/06/2019).

[37] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[38] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[39] Song Han, Huizi Mao, and William Dally. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”. In: Oct. 2016.

[40] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[41] Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: (Dec. 2013).

[42] Sik-Ho Tsang. *Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification)*. 2018. URL: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7 (visited on 05/06/2019).

[43] Laurent Sifre and Stéphane Mallat. "Rigid-Motion Scattering for Texture Classification". In: *CoRR* abs/1403.1687 (2014). arXiv: 1403.1687. URL: http://arxiv.org/abs/1403.1687.

[44] S. J. Pan and Q. Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010), pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191.

[45] Waseem Rawat and Zenghui Wang. "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review". In: *Neural Computation* 29 (June 2017), pp. 1–98. DOI: 10.1162/NECO_a_00990.

[46] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792. URL: http://arxiv.org/abs/1411.1792.

[47] Chuanqi Tan et al. "A Survey on Deep Transfer Learning". In: *CoRR* abs/1808.01974 (2018). arXiv: 1808.01974. URL: http://arxiv.org/abs/1808.01974.

[48] Yichuan Tang. "Deep Learning using Support Vector Machines". In: *CoRR* abs/1306.0239 (2013). arXiv: 1306.0239. URL: http://arxiv.org/abs/1306.0239.

[49] Pedro Marcelino. *Transfer learning from pre-trained models*. 2018. URL: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751 (visited on 05/06/2019).

[50] C. Zach, T. Pock, and H. Bischof. "A duality based approach for realtime tv-l1 optical flow". In: *In Ann. Symp. German Association Patt. Recogn.* 2007, pp. 214–223.

[51] Thomas Brox and Jitendra Malik. "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation". In: *IEEE transactions on pattern analysis and machine intelligence* 33 (Mar. 2011), pp. 500–13. DOI: 10.1109/TPAMI.2010.143.

[52] Aditya Khosla et al. "Novel Dataset for Fine-Grained Image Categorization : Stanford Dogs". In: 2012.

[53] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[54] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[55] M. Oquab et al. “Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. June 2014, pp. 1717–1724. DOI: 10.1109/CVPR.2014.222.

[56] Hossein Azizpour et al. “From Generic to Specific Deep Representations for Visual Recognition”. In: *CoRR* abs/1406.5774 (2014). arXiv: 1406.5774. URL: http://arxiv.org/abs/1406.5774.

[57] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. “UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild”. In: *CoRR* abs/1212.0402 (2012). arXiv: 1212.0402. URL: http://arxiv.org/abs/1212.0402.

[58] Christoph Feichtenhofer. *Convolutional Two-Stream Network Fusion for Video Action Recognition*. Sept. 27, 2016. URL: https://github.com/feichtenhofer/twostreamfusion (visited on 04/10/2019).

[59] NumPy developers. *NumPy*. Scipy.org. 2019. URL: https://www.numpy.org/ (visited on 04/10/2019).

[60] Marshall Copeland et al. *Microsoft Azure: Planning, Deploying, and Managing Your Data Center in the Cloud*. 1st. Berkely, CA, USA: Apress, 2015. ISBN: 1484210441, 9781484210444.

[61] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[62] François Chollet et al. *Keras*. https://keras.io. 2015.

[63] Fredrik Lundh, Alex Clark and contributors. *Pillow*.

[64] Microsoft. *Transfer learning from pre-trained models*. 2019. URL: https://azure.microsoft.com/en-us/services/machine-learning-service/ (visited on 04/04/2019).

[65] Mathew Monfort et al. “Moments in Time Dataset: one million videos for event understanding”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), pp. 1–8. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2019.2901464.

Appendix A

Classification results

This part shows entire contents of classification reports, reporting precision, recall and f1-score for each of the 101 classes and their average values over the 101 classes for both of the GRU-based approaches respectively based on quantized MobileNetV2 and Inception-v3 feature extraction. We also subsequently shows confusion matrices without normalization for both of the GRU-based approaches, along with numbers of correct and incorrect predictions (Figues A.1 and A.2).

Classification report quantized MobileNetV2 feature extraction GRU

```
                    precision    recall  f1-score   support

    ApplyEyeMakeup       0.67      0.64      0.65        44
     ApplyLipstick       0.62      0.78      0.69        32
           Archery       0.62      0.61      0.62        41
      BabyCrawling       0.79      0.97      0.87        35
       BalanceBeam       0.76      0.61      0.68        31
      BandMarching       0.84      0.95      0.89        43
     BaseballPitch       0.95      0.86      0.90        43
        Basketball       0.47      0.91      0.62        35
    BasketballDunk       0.97      1.00      0.99        37
        BenchPress       0.85      0.96      0.90        48
            Biking       0.97      0.97      0.97        38
         Billiards       1.00      1.00      1.00        40
       BlowDryHair       0.76      0.68      0.72        38
    BlowingCandles       0.66      0.94      0.78        33
  BodyWeightSquats       0.19      0.13      0.16        30
           Bowling       0.87      0.93      0.90        43
BoxingPunchingBag       0.92      0.45      0.60        49
    BoxingSpeedBag       0.53      0.81      0.64        37
```

BreastStroke	0.53	0.96	0.68	28
BrushingTeeth	0.31	0.31	0.31	36
CleanAndJerk	0.61	0.70	0.65	33
CliffDiving	0.88	0.97	0.93	39
CricketBowling	0.36	0.39	0.37	36
CricketShot	0.41	0.27	0.32	49
CuttingInKitchen	0.91	0.94	0.93	33
Diving	0.96	1.00	0.98	45
Drumming	0.81	0.87	0.84	45
Fencing	0.71	0.85	0.77	34
FieldHockeyPenalty	0.62	0.45	0.52	40
FloorGymnastics	0.89	0.67	0.76	36
FrisbeeCatch	0.60	0.81	0.69	37
FrontCrawl	0.87	0.35	0.50	37
GolfSwing	0.52	0.87	0.65	39
Haircut	0.45	0.67	0.54	33
HammerThrow	0.83	0.64	0.73	45
Hammering	0.85	0.33	0.48	33
HandStandPushups	0.68	0.68	0.68	28
HandstandWalking	0.08	0.06	0.07	34
HeadMassage	0.47	0.78	0.59	41
HighJump	0.90	0.49	0.63	37
HorseRace	0.89	0.97	0.93	35
HorseRiding	0.96	1.00	0.98	49
HulaHoop	0.62	0.59	0.61	34
IceDancing	0.85	0.98	0.91	46
JavelinThrow	0.77	0.65	0.70	31
JugglingBalls	0.51	0.50	0.51	40
JumpRope	0.11	0.08	0.09	38
JumpingJack	0.79	0.62	0.70	37
Kayaking	0.83	0.69	0.76	36
Knitting	1.00	0.82	0.90	34
LongJump	0.75	0.54	0.63	39
Lunges	0.40	0.46	0.43	37
MilitaryParade	0.88	0.91	0.90	33
Mixing	1.00	0.62	0.77	45
MoppingFloor	0.65	0.71	0.68	34
Nunchucks	0.33	0.20	0.25	35
ParallelBars	0.85	0.92	0.88	37
PizzaTossing	0.48	0.42	0.45	33
PlayingCello	1.00	0.77	0.87	44
PlayingDaf	0.89	0.80	0.85	41
PlayingDhol	0.70	0.88	0.78	49
PlayingFlute	1.00	0.85	0.92	48
PlayingGuitar	1.00	1.00	1.00	43
PlayingPiano	0.80	1.00	0.89	28
PlayingSitar	1.00	0.98	0.99	44

```
       PlayingTabla       0.91      1.00      0.95        31
      PlayingViolin       0.92      0.86      0.89        28
          PoleVault       0.93      1.00      0.96        40
        PommelHorse       0.86      0.51      0.64        35
            PullUps       0.44      0.25      0.32        28
              Punch       0.81      0.87      0.84        39
            PushUps       0.68      0.57      0.62        30
            Rafting       0.89      0.89      0.89        28
RockClimbingIndoor       0.93      1.00      0.96        41
       RopeClimbing       0.72      0.68      0.70        34
             Rowing       0.80      0.92      0.86        36
          SalsaSpin       0.48      0.58      0.53        43
       ShavingBeard       0.53      0.44      0.48        43
            Shotput       0.61      0.59      0.60        46
      SkateBoarding       0.68      0.66      0.67        32
             Skiing       0.81      0.75      0.78        40
             Skijet       1.00      1.00      1.00        28
          SkyDiving       0.97      0.97      0.97        31
     SoccerJuggling       0.43      0.41      0.42        39
      SoccerPenalty       0.81      0.83      0.82        41
         StillRings       0.97      0.94      0.95        32
      SumoWrestling       0.94      1.00      0.97        34
            Surfing       0.94      1.00      0.97        33
              Swing       0.70      0.88      0.78        42
    TableTennisShot       0.93      0.97      0.95        39
             TaiChi       0.74      0.61      0.67        28
        TennisSwing       0.46      0.24      0.32        49
        ThrowDiscus       0.38      0.76      0.50        38
  TrampolineJumping       0.79      0.94      0.86        32
             Typing       1.00      0.91      0.95        43
         UnevenBars       0.70      0.93      0.80        28
  VolleyballSpiking       0.78      0.89      0.83        35
     WalkingWithDog       0.70      0.78      0.74        36
        WallPushups       0.46      0.31      0.37        35
     WritingOnBoard       1.00      0.96      0.98        45
               YoYo       0.80      0.56      0.66        36

        avg / total       0.74      0.73      0.72      3783
```

Classification report Inception-v3 feature extraction GRU

```
                     precision    recall  f1-score   support

      ApplyEyeMakeup       0.86      0.73      0.79        44
       ApplyLipstick       0.69      0.62      0.66        32
             Archery       0.83      0.93      0.87        41
        BabyCrawling       0.92      1.00      0.96        35
         BalanceBeam       0.51      0.61      0.56        31
        BandMarching       0.72      0.98      0.83        43
       BaseballPitch       0.74      0.81      0.78        43
          Basketball       0.54      0.74      0.63        35
      BasketballDunk       0.84      1.00      0.91        37
          BenchPress       0.82      0.83      0.82        48
              Biking       0.92      0.95      0.94        38
           Billiards       1.00      1.00      1.00        40
         BlowDryHair       0.84      0.71      0.77        38
      BlowingCandles       0.89      1.00      0.94        33
    BodyWeightSquats       0.47      0.23      0.31        30
             Bowling       0.87      0.93      0.90        43
  BoxingPunchingBag       0.74      0.80      0.76        49
      BoxingSpeedBag       0.67      0.81      0.73        37
        BreastStroke       0.56      0.71      0.63        28
       BrushingTeeth       0.75      0.50      0.60        36
        CleanAndJerk       0.75      0.73      0.74        33
         CliffDiving       0.90      0.90      0.90        39
      CricketBowling       0.35      0.31      0.33        36
         CricketShot       0.48      0.45      0.46        49
    CuttingInKitchen       0.73      1.00      0.85        33
              Diving       0.94      0.98      0.96        45
            Drumming       0.95      0.91      0.93        45
             Fencing       0.94      0.88      0.91        34
 FieldHockeyPenalty       0.74      0.80      0.77        40
     FloorGymnastics       0.61      0.78      0.68        36
        FrisbeeCatch       0.80      0.76      0.78        37
          FrontCrawl       0.70      0.51      0.59        37
           GolfSwing       0.64      0.59      0.61        39
             Haircut       0.54      0.82      0.65        33
         HammerThrow       0.55      0.69      0.61        45
           Hammering       0.76      0.67      0.71        33
    HandStandPushups       0.72      0.64      0.68        28
    HandstandWalking       0.33      0.21      0.25        34
         HeadMassage       0.87      0.83      0.85        41
            HighJump       0.71      0.46      0.56        37
           HorseRace       0.84      0.91      0.88        35
         HorseRiding       0.94      1.00      0.97        49
            HulaHoop       0.90      0.82      0.86        34
         IceDancing       1.00      0.98      0.99        46
       JavelinThrow       0.58      0.45      0.51        31
      JugglingBalls       0.74      0.88      0.80        40
           JumpRope       0.17      0.05      0.08        38
        JumpingJack       0.67      0.54      0.60        37
           Kayaking       0.88      0.83      0.86        36
           Knitting       1.00      0.82      0.90        34
           LongJump       0.46      0.59      0.52        39
             Lunges       0.71      0.41      0.52        37
     MilitaryParade       0.90      0.82      0.86        33
             Mixing       1.00      0.78      0.88        45
       MoppingFloor       0.67      0.82      0.74        34
          Nunchucks       0.35      0.31      0.33        35
       ParallelBars       0.62      0.97      0.76        37
       PizzaTossing       0.59      0.61      0.60        33
       PlayingCello       1.00      0.68      0.81        44
         PlayingDaf       0.89      1.00      0.94        41
        PlayingDhol       1.00      1.00      1.00        49
       PlayingFlute       0.95      0.85      0.90        48
      PlayingGuitar       1.00      1.00      1.00        43
       PlayingPiano       0.96      0.86      0.91        28
       PlayingSitar       1.00      1.00      1.00        44
       PlayingTabla       0.90      0.84      0.87        31
      PlayingViolin       0.70      1.00      0.82        28
          PoleVault       0.69      0.95      0.80        40
        PommelHorse       0.93      0.74      0.83        35
            PullUps       0.90      0.64      0.75        28
              Punch       1.00      0.87      0.93        39
            PushUps       0.79      0.73      0.76        30
            Rafting       0.89      0.89      0.89        28
RockClimbingIndoor       0.98      0.98      0.98        41
       RopeClimbing       0.65      0.71      0.68        34
             Rowing       0.88      0.78      0.82        36
          SalsaSpin       0.66      0.72      0.69        43
       ShavingBeard       0.56      0.88      0.68        43
            Shotput       0.70      0.50      0.58        46
      SkateBoarding       0.67      0.75      0.71        32
             Skiing       0.81      0.75      0.78        40
             Skijet       0.97      1.00      0.98        28
          SkyDiving       0.97      0.97      0.97        31
     SoccerJuggling       0.48      0.59      0.53        39
      SoccerPenalty       0.94      0.83      0.88        41
          StillRings       0.88      0.72      0.79        32
      SumoWrestling       0.83      0.88      0.86        34
            Surfing       0.87      1.00      0.93        33
              Swing       0.81      0.81      0.81        42
    TableTennisShot       1.00      1.00      1.00        39
```

```
             TaiChi       0.78      0.75      0.76        28
        TennisSwing       0.68      0.43      0.53        49
        ThrowDiscus       0.54      0.71      0.61        38
  TrampolineJumping       0.69      0.97      0.81        32
             Typing       1.00      0.81      0.90        43
         UnevenBars       1.00      0.86      0.92        28
  VolleyballSpiking       0.59      0.69      0.63        35
     WalkingWithDog       0.72      0.86      0.78        36
        WallPushups       0.56      0.43      0.48        35
     WritingOnBoard       0.95      0.91      0.93        45
               YoYo       0.74      0.64      0.69        36

        avg / total       0.77      0.77      0.76      3783
```

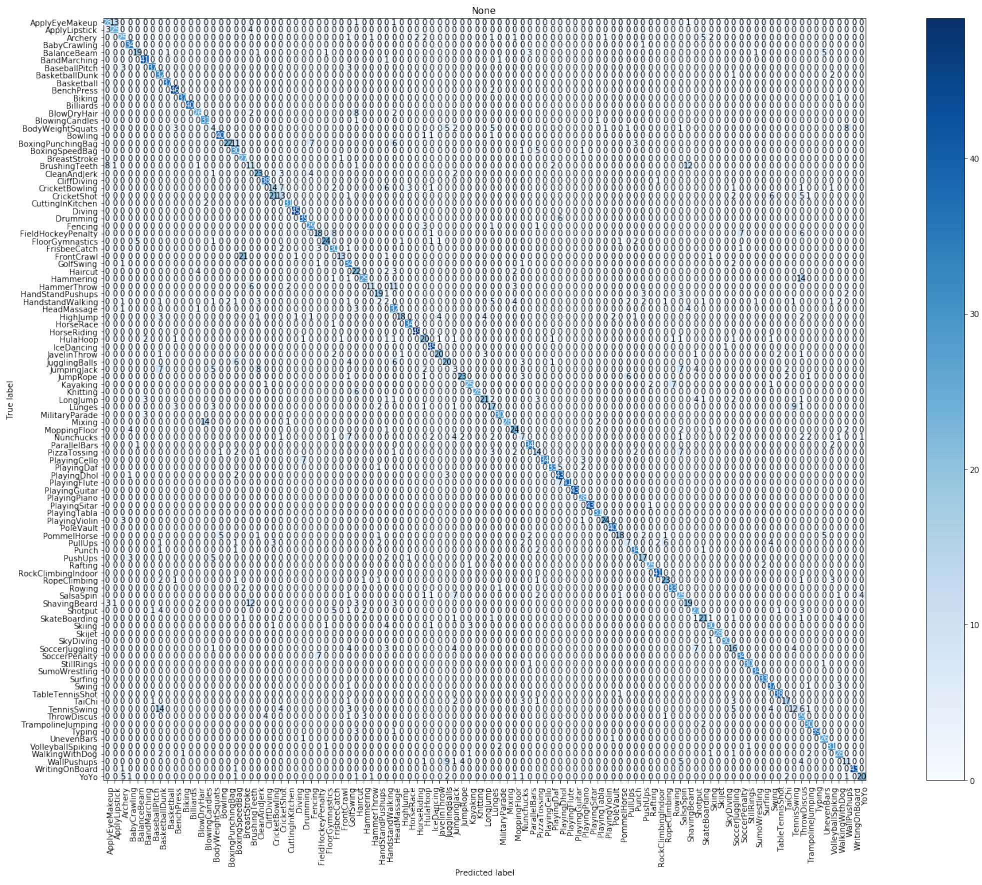


Figure A.1: Confusion matrix, without normalization (quantized MobileNetV2 feature extraction and GRU classifier), along with numbers of correct and incorrect predictions. Color depth indicates number of predictions.

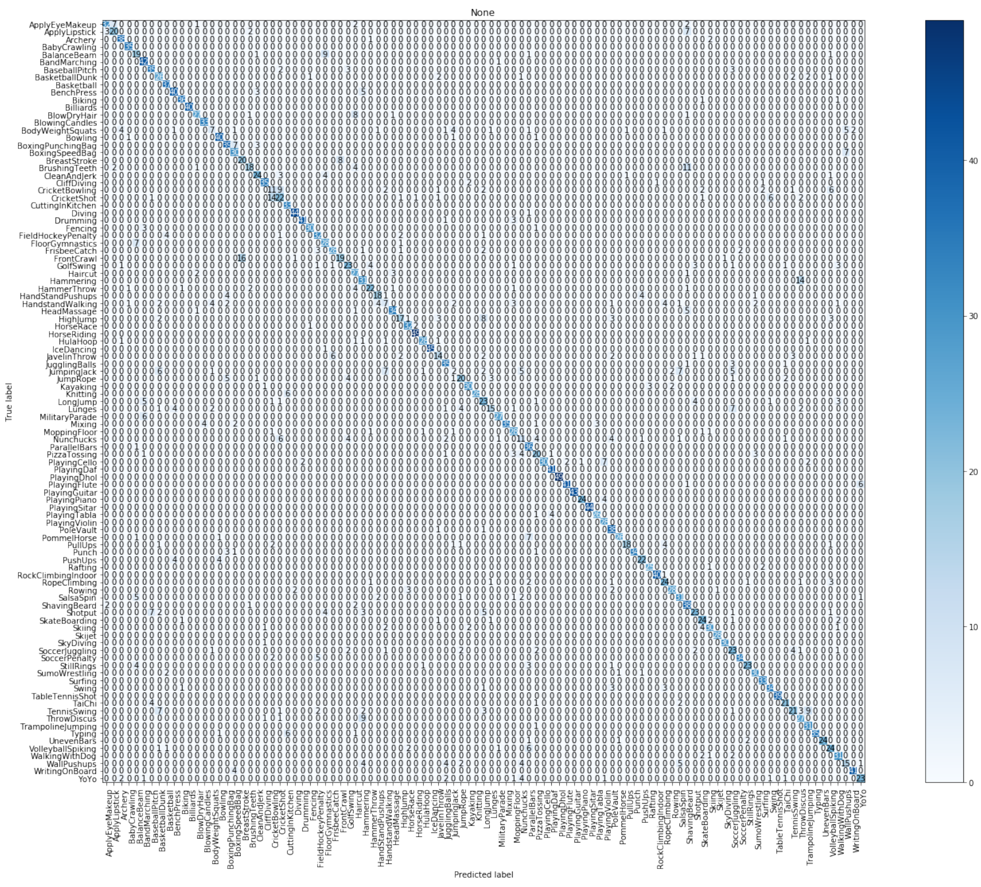


Figure A.2: Confusion matrix, without normalization, (Inception-v3 feature extraction and GRU classifier), along with numbers of correct and incorrect predictions. Color depth indicates number of predictions.

Appendix B

Deployment of the video classification pipeline

The video classification pipeline based on quantized MobileNetV2 feature extraction and GRU classifier is actually deployed on an example of a device with limited computational capabilities and memories: a Raspberry Pi. Knowing the right number of epochs to train the GRU classifier, the latter is re-trained on the whole dataset and saved in Microsoft Azure then downloaded.

The TensorFlow devel docker image tensorflow/tensorflow:nightly-devel is used in order to cross compile and build TensorFlow Lite [22]. The video classification pipeline is developed within a docker container using this image within a local machine. The trained GRU classification model is put together with the quantized MobileNetV2 feature extractor. Finally, files in the container are zipped. Besides, on the Raspberry Pi, a docker container is created and uses the same docker image. The zip file is loaded into this container in order to deploy the video classification pipeline on the Raspberry Pi.