Piotr Ludvig Kopczynski

Video understanding with the Youtube-8M dataset

Master's thesis in MTFYMA
Supervisor: Thiago Guerrera Martins
June 2022

Piotr Ludvig Kopczynski

Video understanding with the Youtube-8M dataset

Master's thesis in MTFYMA
Supervisor: Thiago Guerrera Martins
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Kunnskap for en bedre verden

NTNU

Kunnskap for en bedre verden

Department of Mathematics

TMA4900 - Industriell matematikk, masteroppgave

Video understanding with the Youtube-8M dataset

Author:
Piotr Ludvig Kopczynski

June, 2022

Contents

List of Figures

List of Tables

1 Abstract

In this project, deep learning models were trained on the Youtube-8M dataset, which is a large-scale benchmark for multi-label video classification, and evaluated using the F1-score metric. The trained models used different methods for representing video based on its frames, and a comparison was made between them. The methods used in the project were Recurrent Neural Networks, Transformer based networks, average pooling, and learnable pooling such as Deep Bag of Frames, Net Vectors of Locally Aggregated Descriptors, and Net Fisher Vectors. Experiments with hyperparameter tuning, network architecture, regularization and adding a learnable non-linear unit called Context Gating were performed in order to improve the F1-score of the individual models. The results showed that for sequential models, Recurrent Neural Networks were outperformed by Transformer based models, which again were outperformed by every pooling model except of Deep Bag of Frames, where the model having the highest test F1-score was based on Net Vectors of Locally Aggregated Descriptors.

2 Introduction

The lack of a large labeled video dataset has for a long time been a big obstacle to rapid improvements in video understanding research. To that end, Google's release of the Youtube-8M dataset[33] and making it open source was a large step to kickstart innovation within this field. In addition Google has hosted three different competitions on the kaggle website, that challenged the public to develop classification algorithms which accurately assign video- and segment-level labels [9][31][32].

A video can contain multiple topics not characterized by the uploader. Extracting this information can benefit various applications like video search, video recommendations, video summarization, video content safety, and much more. Furthermore, such research will also help broaden the knowledge about machine learning as new algorithms and architectures are developed. The Youtube-8M dataset describes videos using a numeric representation of all its frames and audio, and is accessible at three levels [16]. In this project, due to storage limitations, only a 400GB subset of the originally 1.2TB frame-level dataset is used, in addition to the whole video-level dataset. The sheer size of the dataset used in this project introduces a minor challenge, which is time and computational resources required for processing.

The goal of this project is to implement and experiment with different types of deep learning models that can learn from the Youtube-8M dataset, and compare their performances.

3 Theory

In the Youtube-8M dataset, a video can have several labels associated with it, making it a multi-label classification problem.

3.1 Multi-label classification

Classification in machine learning is a supervised learning approach where a computer program learns from observing labeled data, and based on that experience can predict labels of new unlabeled data.

Multi-label classification is when a given observation can belong to one or more classes. This is different than the classical multi-class classification where each observation can belong to only one of the available classes.

3.1.1 Multi-hot encoding

Multi-hot encoding is a way of numerically representing an array of labels. The representation consists of a 0 or 1 value for each possible label, 0 meaning that the label is absent and 1 meaning that it is present. As an example let there be 4 possible labels apple, banana, orange and strawberry in this specific order. If a vector contains apple and orange, the multi-hot encoding becomes 1,0,1,0.

3.2 Loss function

A loss function is a function that evaluates the goodness of fit of a machine learning model, by comparing the model prediction with the expected output.

For each task there can be multiple loss functions to choose from. In the case of multi-label classification problems, where the output is multi-hot encoded, one such useful loss function is binary cross-entropy [20].

$$Loss = -\frac{1}{N}\sum y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i), \tag{1}$$

where $\hat{y}_i$ is value number i of the model output, y_i is the corresponding correct target value, and N is the number of values in the model output. Note the logarithm expression in the formula. This could be problematic, but $\hat{y}_i$ is the outcome of a sigmoid function so $\hat{y}_i \in (0,1)$.

3.3 Optimizers

To find the minimum of the previously mentioned loss function, an efficient numerical optimizing algorithm is preferred. The optimizer used on all models used in this project will be described.

3.3.1 Adam

Adam stands for Adaptive Moment Estimation. The algorithm for updating weights is as follows[11].

$$w_t = w_{t-1} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}, \tag{2}$$

where

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \tag{3}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{4}$$

and lastly

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \tag{5}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2, \tag{6}$$

where β_1 and β_2 are scalar numbers called forgetting parameters.

Adam's way of converging to the minimum is described as a ball with both momentum and friction[3].

3.4 Evaluation metric

To evaluate the performance of a machine learning model, it is important to have a suitable evaluation metric. One of many suitable metrics could be the F1-score [23]. Before defining the F1-score, precision and recall are defined.

3.4.1 Precision

Precision can be used as a machine learning metric in itself, but it also takes part in computing the F1-score. Precision is defined as follows.

$$\text{Precision} = \frac{\text{Number of True Positives}}{\text{Number of True Positives} + \text{Number of False Positives}}. \tag{7}$$

What these values mean can be seen in a confusion matrix.

	Prediction outcome	
actual value	True Positive	False Negative
	False Positive	True Negative

Table 1: Confusion matrix

When the prediction outcome and actual value coincide we get that it is true, otherwise it is false. Also when referring to a positive and negative outcome, in terms of binary classification it means a predicted outcome of 1 and 0 respectively.

So precision gives us the percentage of true positive predictions among all positive predictions. A precise model may not find all the true positive outcomes, but the ones that the model does deem to be positive are likely to be true, while a model with low precision may have a lot of positive predictions, but a lot of those predictions are not actually positive.

3.4.2 Recall

Recall can also be used as a machine learning metric and takes part in computing the F1-score. Recall is defined as follows.

$$\text{Recall} = \frac{\text{Number of True Positives}}{\text{Number of True Positives} + \text{Number of False Negatives}}. \tag{8}$$

Recall can be interpreted as measuring how many true positive outcomes did the model find out of all the outcomes that the model deemed to be positive. So a model with high recall is likely to find all the true positive cases, although it may also wrongly identify true negative outcomes as positive, while a model with low recall is not able to find a large part of the true positive outcomes.

3.4.3 F1-score

Now the F1-score combines precision and recall in the following way.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{9}$$

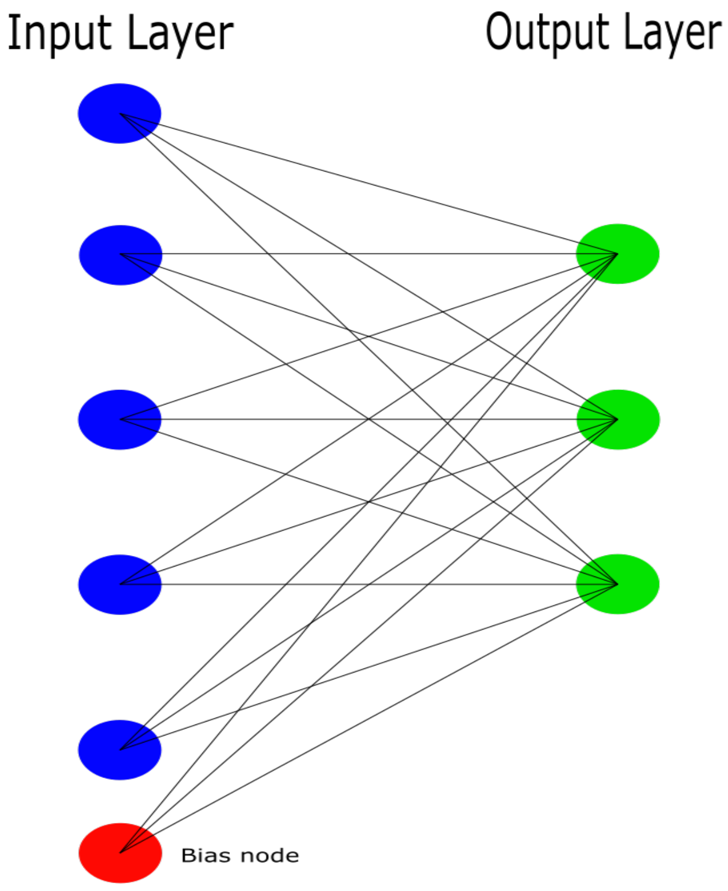
This metric gives equal weight to precision and recall, and is useful when dealing with imbalanced data because it can distinguish between specific types of errors.

3.5 Multinomial logistic neural network

Deep learning[8] is a subfield of machine learning that learns representations from data with an emphasis on learning successive layers of increasingly meaningful representations. These layered representations are learned using models called neural networks.

The introduction of neural network models used in this project starts with an explanation of the general idea of a baseline neural network for classification [37]. A simple neural network structure is presented in figure 1.

Figure 1: Base neural network



Neural network structure with only an input layer with 5 input nodes (blue) and 1 bias node (red), and an output layer with 3 nodes (green).

In figure 1 we see the two main components.

- The input layer: Being where the model receives the input containing n number of features $x_1, x_2, ..., x_n = \mathbf{x}$ of an observation.
- The output layer: Being where the nodes from the input layer are transformed into output, which in this case are the video labels.

The above-mentioned transformation is given by

$$\hat{y}_{ik}(\mathbf{x}_i) = \phi_o(w_0 + w_1 x_{i1} + ... + w_n x_{in}) \quad k \in C, \tag{10}$$

where $\hat{y}_{ik}(\mathbf{x}_i)$ is prediction number i given feature number i belonging to class k given that the output can be one of a number of C classes, ϕ_o is the so called activation function of the output layer that is chosen when designing the network, and lastly the weights $w_0, w_1, ..., w_n$ that are scalar numbers. Note the weight w_0. This is called the bias, which is a scalar number analogous to the intercept in a regression model[38].

An useful activation function for multi-label classification problems is the sigmoid function [19]

$$S(x) = \frac{1}{1 + e^{-x}} \in (0, 1), \tag{11}$$

also shown in the figure below.

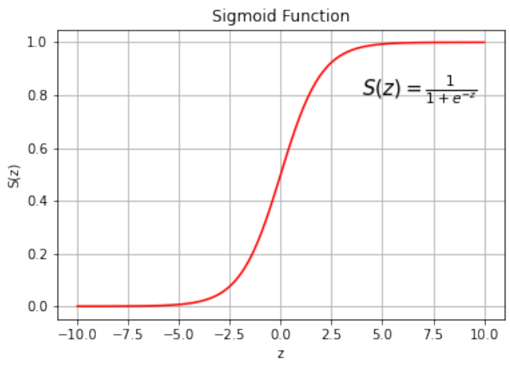


Figure 2: Sigmoid function

The sigmoid function maps input to an output ranging from 0 to 1 that can represent probability of a label being correct. A value > 0.5 represents the presence of a label while a value ≤ 0.5 represents its absence. For small input (< -5), the sigmoid returns a value close to 0, and for large inputs (> 5), it returns values close to 1. This function is often used for binary classification, but since several output values are independent of each other, it is also suitable for multi-label classification.

Having no other layers except the input and output layers, makes this a linear model, because the output always depends on the product of the inputs and their weights. Linear models are inflexible models, which means that they can not identify nonlinear dependencies between the input data, and the output. Consequently, an inflexible model will likely underfit if used on this type of data. The complexity of this model can be increased by adding a variety of hidden layers using a variety of activation functions.

3.6 Recurrent Networks

Recurrent neural networks [18], RNNs for short, are a family of neural networks for processing sequential data. The key idea in RNNs is that the neurons maintain an internal state which is updated at each timestep t as a sequence of inputs $[x_1, ..., x_t, ..., x_T]$ is processed. Structurally, a RNN is divided into cells, where one cell processes one sequence. During processing of a sequence, a RNN cell computes a cell state at time t denoted as h_t given by

$$h_t = f_W(x_t, h_{t-1}), \tag{12}$$

where f_W i a function parameterized by a set of trainable weights W. These weights are what the RNN is trying to learn over the course of training using a variation of the backpropagation algorithm [12] called back propagation through time [13], BPTT for short. Given this, the output of a RNN cell at a timestep t is given as a function of the current input x_t, and the past memory h_{t-1}.

3.6.1 Gated RNNs

In a basic RNN, it is easy for the internal state which remembers past information, to forget information after a couple of timesteps, resulting in short memory. In addition the computations inside a basic RNN include many matrix multiplications when computing how to change the internal state, in other words when computing the gradient with respect to the internal state. This introduces two problems. The first one is that there being many values larger than 1 present in the multiplications, causes the exploding gradients problem [5] which makes the optimization difficult. The second problem is the opposite. There being many values less than 1 may lead to vanishing gradients [5], which makes it harder to backpropagate the error from the loss function back to the distant past.

To make the RNNs less susceptible to these problems and more robust, a more complex recurrent unit is introduced by using gated cells. The gates consist of a standard neural network layer using the sigmoid activation function and a pointwise multiplication. The function of these gates is to selectively add or remove information from the cell state h_t. Some common and effective sequence models, used in practical applications are called gated RNNs [18][8],[25]. These include the long short-term memory(LSTM) networks and networks based on the gated recurrent unit(GRU). The LSTM uses three gates inside a recurrent cell:

- An input gate that decides what information will be stored in the cell state by filtering information from the current input x_t.
- A forget gate that selects which information should be kept or discarded from the previous cell state.
- Lastly, an output gate that controls what information is encoded in the cell state and passed to the next timestep, and what the output of the cell will be at a given timestep.

While the GRU uses two gates:

- A reset gate that decides how much of the previous cell state should be kept.
- An update gate that decides whether the cell state should be updated using the current timestep.

Summarizing, the gated RNNs use a more complex neuron structure that lets them combat the gradient problems, and better maintain long term dependencies in the data by utilizing special gates that control the flow of information.

3.7 Transformers

Transformer-based models [8],[36] are another type of sequential models that combat some of the limitations of RNNs which are:

- The encoding bottleneck in the form of information possibly being lost when a lot of data is condensed into a representation that can be used by a RNN for prediction.
- Slow training speed due to the requirement of information being processed sequentially, which is inefficient on modern GPU hardware.
- A memory that does not scale well when the length of sequences are in the thousands.

Transformers have been recently introduced in [30], and have been overtaking RNNs across most natural language processing tasks. The architecture was originally developed as a sequence-to-sequence [7] model, but can be modified for video classification.

One of the key elements of Transformers is that they remove the sequential processing of the input, by using positional embedding. Positional embedding preserves the position information by computing an embedding that captures the positional information, and adding it to the input.

Transformers use the concept of self-attention, which reflects the ability to take an input, identifying which parts to attend to and extract those features with high attention. This is conceptually similar to a search. For example, when searching for a video, we type a query Q into the YouTube search bar. Each video in the YouTube database has key information K about the video, which can for example be the title of the video. To search for a video using the query Q, one can compute the similarity between the query and the keys. This similarity measure, is then used to extract the information that is searched for, denoted as the value V, which is the video itself.

Since the model has some notion of position from the input due to the positional embedding, the task now is to figure out what in the input to attend to. For this, the idea of self attention is used. Transformers create three new unique transformations of the embedded input, which are the query Q, key K, and value V. This is done by making three copies of the positional embedding, and multiplying each with a separate and different linear layer.

The next task is to compute the attention weighting, which represents how much attention should be given to certain features. This is done in the same way as in the mentioned YouTube search example. By computing the similarity between Q and K. This similarity, called attention weighting, is computed by using the dot product

$$\sigma(\frac{Q \cdot K^T}{\sqrt{d_k}}), \tag{13}$$

and scaling it down as in [30] using $\sqrt{d_k}$, where d_k is the dimension of K, and σ is the softmax function [48]. This weighting matrix is then used to extract features from V with high attention by

$$Attention(Q, K, V) = \sigma(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V. \tag{14}$$

This result reflects the features that correspond to high attention. These operations form what is called a self attention-head. A neural network using the transformer encoder can have multiple self attention-heads that attend to different parts of the input.

3.8 Pooling

The previously explained RNN and Transformer models capture the temporal structure of a video when extracting features. The motivation for the models that will be discussed next is the hypothesis that it is not necessary for a good classification performance to treat the video as a sequence if all relevant information about the video labels rely on the static visual and audio cues. The focus of these models is on capturing the distribution of features in the video, without including information about their temporal ordering. In deep learning, pooling [4] describes methods for down sampling feature maps by summarizing the presence of features in the feature map. An example is average pooling, which was used in the process of making the video-level dataset. The pooling was done by adding correspondent features in all frames of a video, and dividing these sums by the number of frames in that video.

3.8.1 Learnable pooling models

Unlike, average pooling, the models presented next learn a pooling method to create a video representation based on its frames.

3.8.2 NetVLAD - Net Vectors of Locally Aggregated Descriptors

NetVLAD is based on Vector of Locally Aggregated Descriptors[34] which is a descriptor pooling method used in image classification, where the term descriptor refers to the image features. This is a clustering based method that captures information about the statistics of local descriptors aggregated over an image or also in this case audio, and stores the difference vector between the descriptors and their corresponding cluster centres.

We have N D-dimensional frame descriptors x_i as input, and K cluster centres c_k as VLAD parameters. For convenience, we denote the output of VLAD as a matrix $\mathbf{V}$ with dimensions $K \times D$. This matrix is converted to a vector and after normalization, is used as the video representation. Element v_j, k of $\mathbf{V}$ is given as follows

$$v_{j,k} = \sum_{i=1}^{N} a_k(x_i)(x_{i,j} - c_{k,j}), \tag{15}$$

where $a_k(x_i)$ denotes the membership of descriptor x_i to the k-th cluster, meaning that it is equal to 1 if cluster c_k is the closest cluster to x_i and 0 otherwise. This means that each column of $\mathbf{V}$ stores the sum of residuals $(x_i - c_k)$ of descriptors which are assigned to cluster c_k. $\mathbf{V}$ is then L2-normalized [46] column wise, before being converted to a vector and L2-normalized in its entirety.

The NetVLAD architecture [27] reproduces the VLAD encoding in a differentiable manner, such that the clusters can be learned using the backpropagation algorithm[12]. This is done by by writing $a_k(x_i)$ as a soft assignment

$$a_k(x_i) = \frac{e^{w_k^T x_i + b_k}}{\sum_{l=1}^{K} e^{w_l^T x_i + b_l}}, \tag{16}$$

where w and b are learnable parameters. This means that instead of being either 0 or 1, $a_k(x_i)$ is now a number between 0 and 1 proportional to how close the descriptor x_i is to cluster c_k.

3.8.3 DBoF - Deep Bag of Frames

The Deep Bag of Frames model is based on the Bag of Words [44] model, which is a model used for extracting features from text by describing the occurrence of certain words within a text. It involves a vocabulary of known words, and a measure of the presence of these words. The motivation for this type of model is the idea that similar texts have similar contents. So we can use the content of a text to check the similarity to other known texts. One downside with this model is that any information about the structure of the text is discarded since the model is not concerned where in the text known words appear, but only about if they actually appear, hence use of "bag" in the name Bag of Words.

This method has also been extended to problems like image classification[26] where instead of searching for specific words in text, a set of key visual features or visual words is defined. This set is used to recognize how often each visual feature is present in an image, and to classify the image based on that.

For this classification task the model is called Deep Bag of Frames [33][1]. In classical Bag of Words and Bag of Visual Words approaches, the set of key features or the "bag" of features were usually found using k-means clustering and a hard assignment to cluster c_k, denoted as $a_k(x_i)$, where x_i is the i-th D-dimensional frame descriptor. The Bag of Words representation is then written as

$$BoW(k) = \sum_{l=1}^{N} a_k(x_i). \tag{17}$$

Since the DBoF approach is used as a Deep Learning model, the representation needs to be differentiable such that it can be learned using the backpropagation algorithm. This is done the same way as for NetVLAD, by defining $a_k(x_i)$ as a soft assignment of descriptors to clusters given by equation (16).

3.8.4 NetFV - Net Fisher Vectors

NetFV is based on the Fisher Vector encoding [21][15] which is an extension of the Bag of Words representation, and stems from the Fisher Kernel[29], which is a function from statistical classification used for measuring the similarity between objects. In image classification, the encoding represents how the distribution of features of an image differs from the distribution fitted to the features of all training images.

The Fisher Vector encoding, which we denote as ϕ, of a set of descriptors x_i is based on fitting a parametric generative model to the feature, and then encoding the derivatives of the log-likelihood of the model with respect to its parameters. In the case where the parametric generative model is the Gaussian Mixture Model[24] with diagonal covariances, the representation of the average first and second order differences between the features and the Gaussian Mixture Model centres becomes

$$\Phi_k^{(1)} = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} a_k(x_i) \left(\frac{x_i - \mu_k}{\sigma_k} \right), \quad \Phi_k^{(2)} = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} a_k(x_i) \left(\frac{(x_i - \mu_k)^2}{\sigma_k^2} - 1 \right), \tag{18}$$

where $k \in [1, K]$, π_k are the mixture weights, μ_k are the mixture means and σ_k are the mixture diagonal covariances of the Gaussian Mixture Model. $a_k(x_i)$ is again the soft assignment given by (16). The Fisher Vector encoding is represented by stacking the differences as follows

$$\phi = \left[\Phi_1^{(1)}, \Phi_1^{(2)}, ..., \Phi_K^{(1)}, \Phi_K^{(2)} \right]. \tag{19}$$

The NetFV method[1] imitates the Fisher Vector encoding by representing the first- and second-order statistics in matrix form as

$$\Phi_{NetFV}^{(1)}(j,k) = \sum_{i=1}^{N} a_k(x_i) \left(\frac{x_{i,j} - c_{k,j}}{\sigma_{k,j}} \right), \quad \Phi_{NetFV}^{(2)} = \sum_{i=1}^{N} a_k(x_i) \left(\frac{(x_{i,j} - c_{k,j})^2}{\sigma_{k,j}^2} - 1, \right) \tag{20}$$

where σ_k is now the diagonal covariances of cluster c_k.

3.9 Context Gating

As mentioned, gates in RNNs are used to selectively pick the information from an input. The winning team [1] of the first Youtube-8M understanding challenge [9] suggested adding this gating mechanism to pooling methods. Context Gating is used to re-weight the output features of the pooling methods and the output labels, such that the resulting features and labels represent the subset of objects and events that are most relevant to the context of the video. In [1] an example is mentioned where, in a skiing video, the network activations corresponding to tree features may be high. But since trees is not an important feature based on the context of the video, the visual network activations of trees are down-weighed by Context Gating. Another motivation for Context Gating is that it can be used for modeling biases in label annotations by creating dependencies among output labels when applied after the classification layer of the network.

In short, Context Gating transforms its input $X \in \mathbb{R}^n$ into a new representation Y given by

$$Y = \sigma(WX + b) \circ X, \tag{21}$$

Where σ is the element-wise sigmoid activation, $\circ$ is element-wise multiplication, $W \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$ are trainable parameters. Consequently, $\sigma(WX + b)$ represents a set of learnable gates applied to the dimensions of the input feature X. This transformation introduces non-linear interactions among activations of the input representations, and reevaluate the strengths of different activations of the input through a self-gating mechanism.

3.10 Regularization

Deep learning [8] can be seen as curve fitting, and a curve fitting model needs to be trained on a dense sampling of its input space to perform well. This means that one of the best ways to improve a deep learning model is to train it on more data or better data. But often it is not possible to obtain more data, and while excessive training on the same dataset can increase training accuracy, it happens often at the expense of a lower testing accuracy. This is because the network essentially memorizes the training cases, which hinders its ability to generalize to new cases. This phenomenon is called overfitting.

The term regularization [8],[14] in deep learning refers to a set of techniques that actively impede the model's ability to fit perfectly to the training data, preventing overfitting. In the following subsections, the regularization techniques applied in this project are presented.

3.10.1 Early Stopping

A practical and commonly used form of regularization is early stopping. This method requires a separate subset of the dataset called a validation set. The model does not use the validation set for learning, but instead during training of a deep learning model, the loss on the validation set is computed after a predetermined number of epochs. Early stopping stops the training if this validation loss stops decreasing.

3.10.2 Dropout

Dropout is one of the most commonly used regularization techniques for neural networks. Large deep learning models tend to be so overparameterized that imposing constraints, like for example L^2 regularization [14], on weight values has low impact on model performance and generalization. In such cases, dropout is a preferred regularization technique.

Dropout applied to a neural network layer consists of randomly dropping out a number of the output features of a neural network layer during training at a rate called the dropout rate. At test time, nothing is dropped out, but instead the layer's output is scaled down by a factor equal to the dropout rate. This is done to balance for the fact that more units are active than at training time. During training, neurons of a layer may change in a way to fix the mistakes of a preceding layer. This may lead to co-dependencies between layers that do not generalize to unseen data. Dropout effectively adds noise to the training process, making it harder for the model to learn these co-dependencies, and thus making the model more robust.

4 Data

The dataset used for this project is the Youtube-8M dataset, which is a large-scale benchmark for general multi-label video classification. The dataset contains more than six million Youtube videos with high-quality machine-generated annotations from a vocabulary of 3862 visually identifiable labels. On the official website there are three versions of the dataset where the features are either frame-level, segment-rated frame-level or video-level. The original size of the video dataset would be impractical to process, so before it was published, there was a substantial amount of preprocessing done [33]. Ultimately, the frame image information has been generalized to 1024 numeric features per frame, and similarly the audio information was generalized to 128 numeric features per frame, and the frames were sampled at a rate of one frame per second of a video. Each video contains a varying amount of labels, averaging to around 3 per video as seen in tables 2,3.

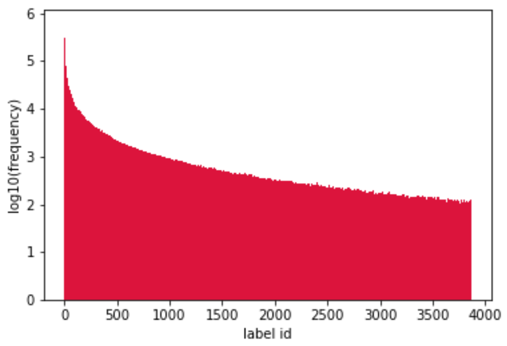
In this project, both the video-level and frame-level datasets are utilized, where video-level means that videos in the dataset are represented by an average of its frames(average pooling), while the frame-level dataset contains videos represented by a list of frames. On the source website for the dataset [16], three subsets of the dataset are available for download: a train set, a validation set and a test set. In addition a vocabulary mapping label id's in the dataset to their respective names and information is available. The target labels for the test set have not been published so for this project only the train set and validation set were used.

4.1 Video-Level Dataset

For video-level modeling, the downloaded validation set was used as the test set, while the train set was split such that 80% of the data was used for training and the rest for validation.

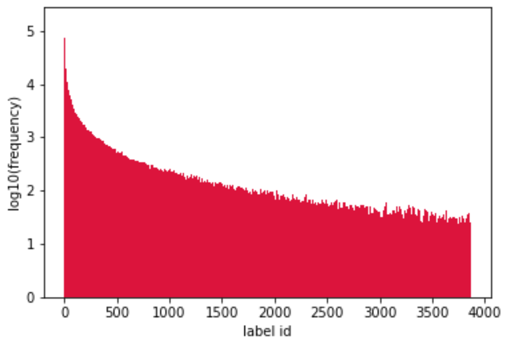
No extensive exploratory data analysis was performed because it was judged to be unnecessary given the nature, quality and size of the dataset. When making the training and validation splits, the tfrecord [45] files that make up the downloaded training dataset were shuffled before splitting into a training part and a validation part in case the downloaded data had an unintended distribution of labels across videos. The following histograms show the distribution of labels in the video-level train, validation and test sets.

Figure 3: Frequency-barplot for the video-level train set



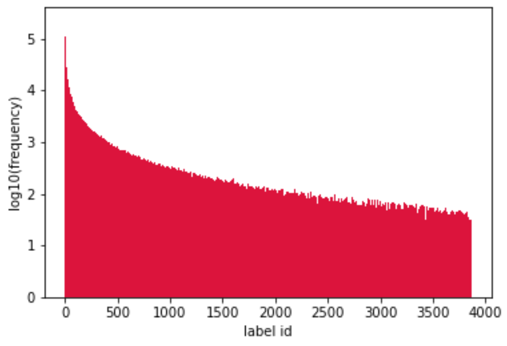
Bar plot featuring label frequencies of the video-level train set, transformed using logarithm with base 10, against label id's.

Figure 4: Frequency-barplot for the video-level validation set



Bar plot featuring label frequencies of the video-level validation set, transformed using logarithm with base 10, against label id's.

Figure 5: Frequency-barplot for the video-level test set



Bar plot featuring label frequencies of the video-level test set, transformed using logarithm with base 10, against label id's.

Table 2: Video-Level Dataset Statistics

statistic	**train dataset**	**validate dataset**	**test dataset**
video count	3888919	3111135	777784
max frequency	630259	158029	225529
min frequency	98	17	26
average label count	3	3	3

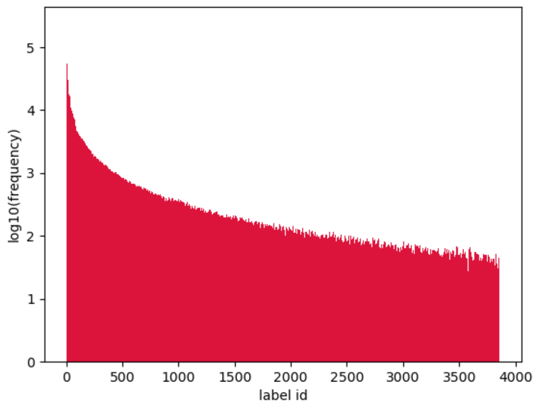
Various video-level dataset statistics.

Luckily, all possible labels appear in all of the three video-level dataset-splits and the distributions seems to coincide well across splits as well. Although, notice that the distribution of labels is by no means equal. A large difference in frequency is observed between the most and least frequent labels, meaning that this is an imbalanced dataset. For example the most frequent label, being "Game", appears in 630259 videos in the training dataset, while the least frequent label "Cylinder" appears only in 98 videos.

4.2 Frame-Level Dataset

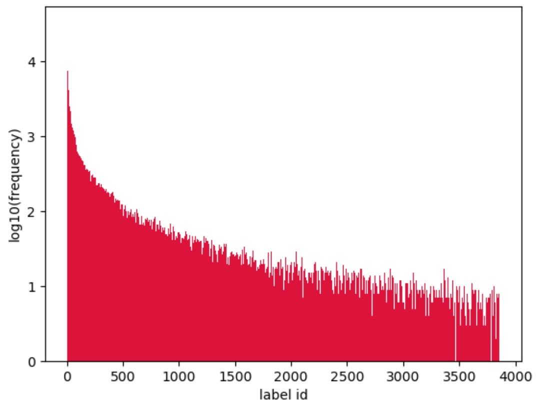
For frame-level modeling, due to storage limitations, only 30% of the frame-level dataset was used. Since the video data was split into 3844 tfrecord files containing a varying number of videos each, it was difficult to modify the distribution of video labels contained in the chosen subset of data. As mentioned, figure 5 showcases the frequency of labels in the appointed test set for video-level models, which is originally the downloaded validation set. It can be seen that it has a similar distribution as the train set showed in 3. Because of this, the downloaded validation set was used as part of the training set. The rest of the available storage space was filled by 10% of the training dataset. This chunk was then randomly distributed such that in total, 80% of data was used for training, 10% was used for validation, and lastly 10% was used for testing. The distributions of these sets are shown in 6,7,8.

Figure 6: Frequency-barplot for the frame-level train split



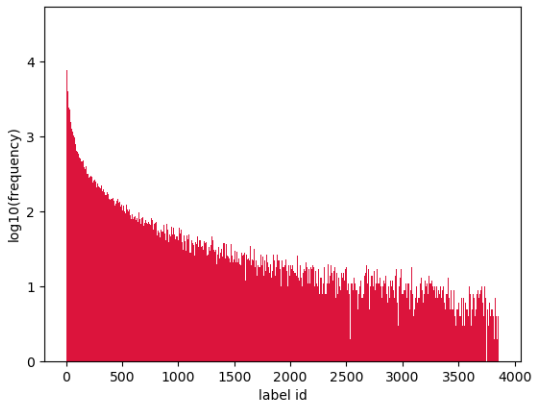
Bar plot featuring label frequencies of the frame-level train dataset, using logarithm transformation with base 10, against label id's.

Figure 7: Frequency-barplot for the frame-level validation split



Bar plot featuring label frequencies of the frame-level validation dataset, using logarithm transformation with base 10, against label id's.

Figure 8: Frequency-barplot for the frame-level test split



Bar plot featuring label frequencies of the frame-level test dataset, using logarithm transformation with base 10, against label id's.

Table 3: Frame-Level Dataset Statistics

statistic	**train dataset**	**validate dataset**	**test dataset**
video count	1181805	159991	160908
max frequency	239528	32517	32416
min frequency	28	0	1
absent labels	None	Cadillac CTS, Dream Club	None
average label count	3	3	3
average frame count	230	230	230

Various frame-level dataset statistics.

4.3 Preprocessing

The datasets are stored in the tfrecord format [45], so the tensorflow dataset API [41] was used to parse the data into the correct structure. As 8 bit quantization [6] was used to compress the visual and audio features in the frame-level dataset, a dequantization function had to be implemented and applied on the features of the frame-level dataset. Furthermore, since we are performing multi label classification, the labels were multi-hot encoded.

The videos are of varying lengths, which can be a problem as most neural networks require input of uniform shape. Since RNN implementations in tensorflow support masking [39], the parsing process included the option for the data to be padded with zeroes such that all videos had an equal length of 300 frames. For learnable pooling, models implemented in this project do not support masking, so instead, data augmentation in the form of random sampling with replacement was used during training such that each video would have a slightly different representation in each epoch. Although, it was made sure that during evaluation, all frames of the videos would appear at least once in this representation.

5 Method

5.1 Model implementation details

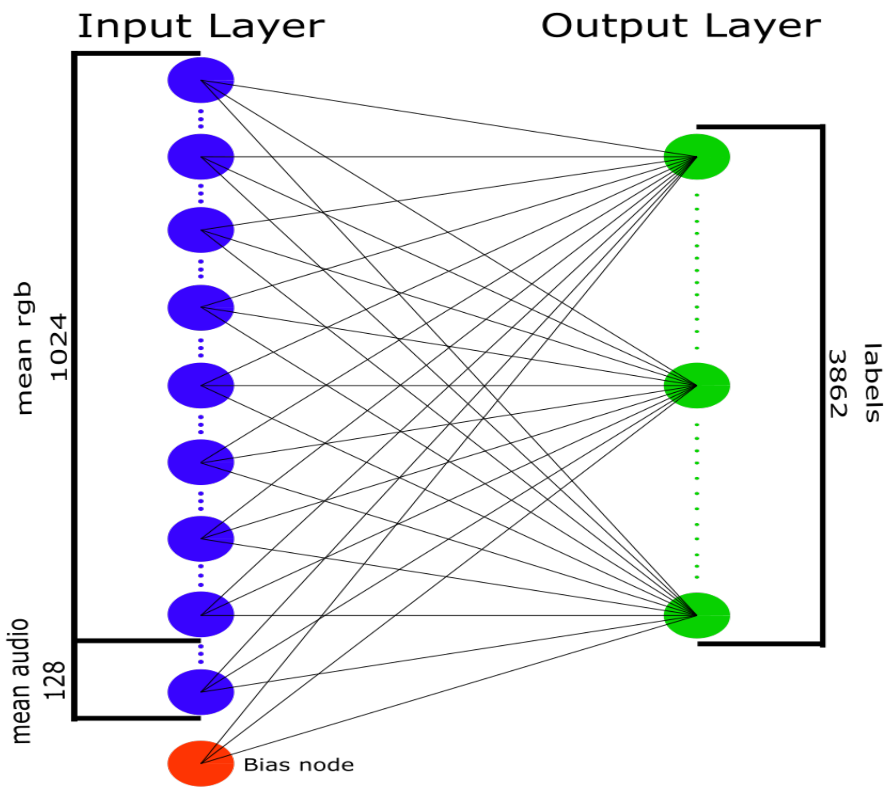
All models were implemented in python using the keras functional API [43] in tensorflow 2, and the computations were ran using NTNU's computational sever markov [17]. The sole optimizer used for all models was the Adam optimizer and the common batch size was 128. Training performance was monitored using tensorboard [40]. Every model used early stopping that monitors the loss on the validation set during training. A model was forced to stop if for 3 epochs, the validation loss did not decrease by more than 0.000001.

The RNN and learnable pooling network architectures used in this project were inspired by the various works previously done with this dataset: [33],[1],[10],[35],[22]. As the goal of this project is to compare the performances of various models, and the time being limited, extensive hyperparameter optimization was omitted by recreating some of the successful hyperparameter combinations used in previous works with this dataset.

5.1.1 Average pooling models

The average pooling models were fitted on both the whole video-level dataset, and the frame-level dataset subset. The choices for learning rate and batch size were based on project work during the previous semester. These were a batch size of 128 and a learning rate of 0.0001. The work started by fitting a logistic regression model on the datasets. The simple architecture is shown in figure 9.

Figure 9: Logistic network architecture



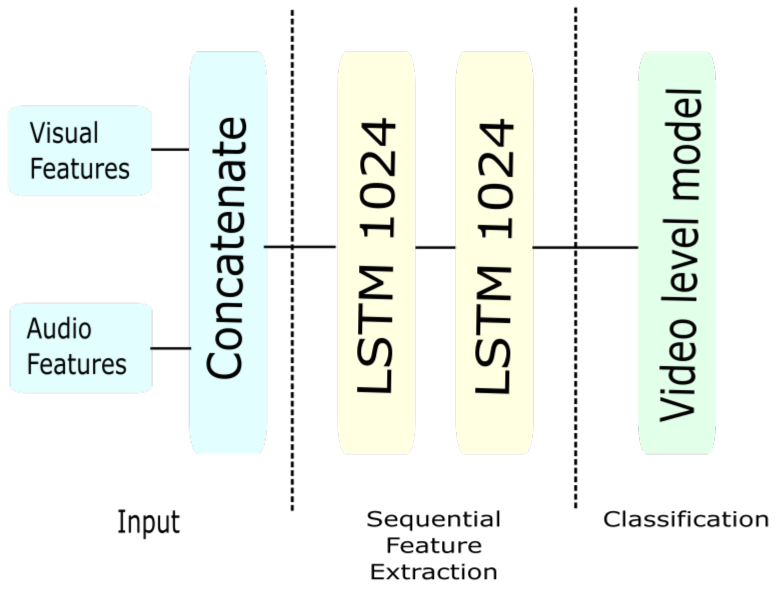
Simplified illustration of the logistic regression model.

Afterwards, experimentation with adding a varying amount of hidden layers of varying widths was done. These layers were the keras implementation of dense layers, and ReLU(Rectified Linear Units) [2] was used as the activation function. The reason for using the ReLU activation in all dense layers is that it is simple, fast and empirically seems to work well in most situations. As the models presented in the following subsections create different types of video representations, they would use one of the models from this section as a video-level module for final classification, to make them comparable. The model chosen based on its runtime and performance was a model consisting of two dense layers with ReLU activations and 2048 and 1024 units respectively, followed by a dense layer with sigmoid activation function, and number of units equivalent to the number of labels in the vocabulary.

5.1.2 Sequential Models

The work started with the implementation of RNN architectures with only one LSTM layer with a few units per cell. After noticing that the RNN models were quite slow, the strategy changed to implementing the best performing RNN architecture from previous works, which is shown in figure 10.

Figure 10: LSTM model architecture

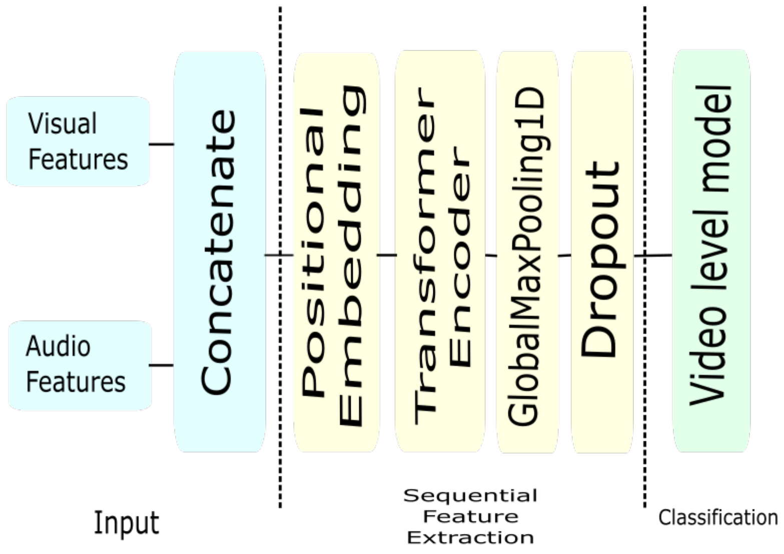


RNN architecture using LSTM layers.

This project uses the built in keras implementations of LSTM and GRU layers.

For models using the transformer encoder, two custom layers had to be implemented. A positional embedding layer and a transformer encoder layer. This was done by following chapter 11.4 in [8] and [28]. The architecture can be seen in 11.

Figure 11: Transformer model architecture



Neural network architecture using the transformer encoder.

Since the TransformerEncoder layer returns full sequences, a global pooling layer was used to reduce each sequence to a single vector for classification. A dropout layer with a dropout rate of 0.35 was included for better generalization of the model. The hyperparameters tuned for this model were the number of units in the dense layers inside the transformer encoder, and the number of self-attention heads. While tuning the transformer models, the video classification module was excluded, and only 2 epochs were trained to save computation time.

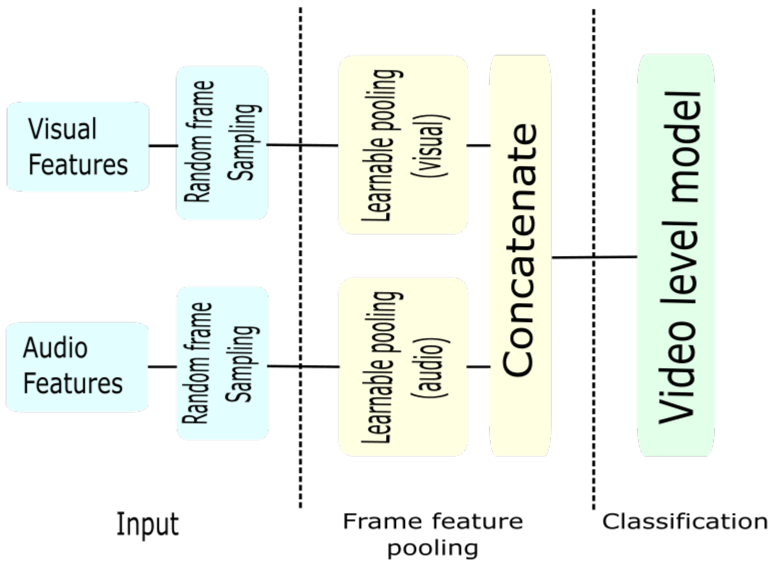
The mentioned sequential models were expected to require a lot of time to train. To speed up the learning process, an initial learning rate of 0.001 was used that is decreased exponentially [42] with the factor of 0.95 every 1500000 samples. In addition to early stopping, a limit was set such that a single model could not train more than five days. This decision was made so other models could be explored in the limited time for this project.

5.1.3 Learnable Pooling models

The learnable pooling models were implemented based on the tensorflow 1 implementation in [1]. Since this project uses tensorflow 2, and the functional API, custom keras layer implementations of DBoF and NetVLAD, NetFV were used from the open source project [49].

The hyperparamenter tuned for these models is the number of clusters. A visual example of the network architecture is shown in figure 12.

Figure 12: Learnable pooling architecture



Example of a two stream learnable pooling architecture.

To speed up the learning process, an initial learning rate of 0.001 is used that is decreased exponentially [42] with the factor of 0.95 every 1500000 samples.

5.2 Additional experiments

The following experiments were performed in addition to model-specific hyperparameter tuning.

5.2.1 Average pooling on more data

Since the video-level dataset is the result of average pooling applied to the whole frame-level dataset, average pooling models with similar architectures were trained on both the frame-level and the video-level datasets to observe the difference in performance resulting in going from training on all videos to training on around 30% of the videos.

5.2.2 Dropout

Observing that some of the average pooling models showed signs of overfitting, by having a considerably higher training F1-score that test F1-score, regularization in the form of dropout was tested. In this experiment, dropout layers were added after each dense layer, except the last classification layer.

5.2.3 Splitting visual and audio features

As audio and visual features are intuitively quite different, it would be reasonable to attempt training models where the video representation was learned on the visual and audio features separately, before being concatenated into one representation. The term ”two stream” will be used from now on to address the case when visual and audio features are separated. These experiments were performed on the average and learnable pooling models. They were not performed on RNN and Transformer models because of the lengthy training time of such models. In [1], better results were attained by concatenating the visual and audio features before training so this was done in this project as well.

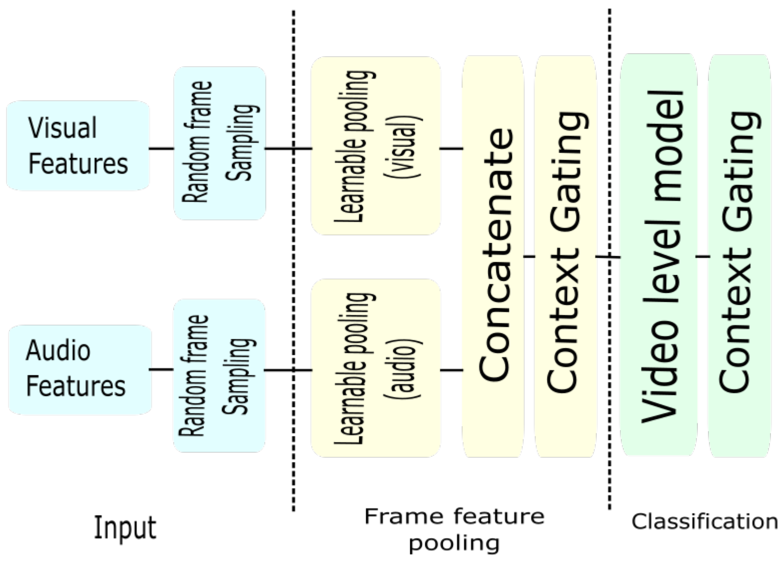
5.2.4 Gradient clipping

The first attempt at training a RNN model using GRU layers yielded a performance that decreased with every epoch. This problem was suspected to be caused by exploding gradients [5]. An attempt to attended this problem was made by applying gradient clipping [47] to the optimizer, which is a method that scales down the gradient during training in the cases where it becomes too large. The clip-value used for gradient clipping was 5.

5.2.5 Context Gating

As suggested in [1], context gating can have a positive impact on the performance of pooling models. In this project, average and learnable pooling models were trained with and without context gating to see if this indeed is the case. Context Gating was implemented as a custom layer. For this the open source code from [49] was used. The Context Gating layer was applied after the pooling and classification modules of average and learnable pooling models. An example of this is shown in figure 13.

Figure 13: Learnable pooling architecture with context gating



Example of a two stream learnable pooling architecture with context gating.

5.3 Best model

The best performing model was designed by combining the best dense layer configuration from average pooling models for video classification, with the best sequential or learnable pooling method for video representation.

6 Results

Table 4: Average pooling results on the video level dataset

hidden layer units	epochs	minutes per epoch	F1-train	F1-test
0	35	10	0.728	0.718
576	14	5	0.787	0.770
576-288	13	4	0.782	0.765
2304	9	12	0.814	0.778
2304-1152	8	9	0.823	0.776
3862	8	22	0.823	**0.780**
3862-1931	7	21	0.841	0.778
6000	8	36	0.837	**0.780**
6000-3000	6	37	0.850	0.779
2304-1152-576	7	8	0.813	0.773
2304-4608	7	29	0.850	0.777
2304-4608-2304	6	25	0.841	0.777
2304-4608-9216	6	76	0.877	0.772
1152-1152-1152-1152	8	9	0.803	0.772

Experiments adding hidden dense layers to average pooling models trained on the video-level dataset.

Table 5: Two stream average pooling results on the video level dataset

hidden layer units	epochs	minutes per epoch	F1-train	F1-test
visual:2048 audio:256	8	9	0.815	0.776
visual:2048-4096 audio:256-512	7	28	0.841	**0.777**
visual:1024-1024-1024 audio:128-128-128	8	8	0.806	0.774

Experiments adding hidden dense layers to average pooling models trained on the video level dataset, where the layers learn from visual and audio features separately.

Table 6: Average pooling results on the frame-level dataset

hidden layer units	**gating**	**epochs**	**minutes per epoch**	**F1-train**	**F1-test**
0	No	29	7	0.730	0.708
0	Yes	18	18	0.806	0.760
2304-1152	No	8	9	0.823	0.761
2304-1152	Yes	8	18	0.814	0.761
3862	No	8	9	0.819	0.766
3862	Yes	10	24	0.827	**0.768**
2304-4608	No	7	29	0.850	0.763
2304-4608	Yes	7	28	0.822	0.764
6000-3000	No	6	14	0.841	0.763
6000-3000	Yes	7	31	0.834	0.763
2304-4608-9216	No	5	52	0.834	0.763
2304-4608-9216	Yes	6	49	0.827	0.759

Experiments adding hidden dense layers and Context Gating to average pooling models on the frame-level dataset.

Table 7: Average pooling results with dropout on the frame level dataset

hidden layer units	**gating**	**dropout rate**	**epochs**	**minutes per epoch**	**F1-train**	**F1-test**
2304-4608-9216	Yes	0.25	9	39	0.812	0.770
2304-4608-9216	No	0.35	13	43	0.809	**0.777**
6000-3000	Yes	0.25	9	19	0.820	0.772
6000-3000	No	0.35	9	17	0.805	**0.777**

Results of adding dropout layers to average pooling models trained on the frame-level dataset, that had a high training F1-score.

Table 8: LSTM model results

LSTM layers	layer units	epochs	minutes per epoch	F1-train	F1-test
1	200	4	101	0.640	0.647
1	500	5	487	0.680	0.687
2	1024	5	1387	0.760	**0.746**

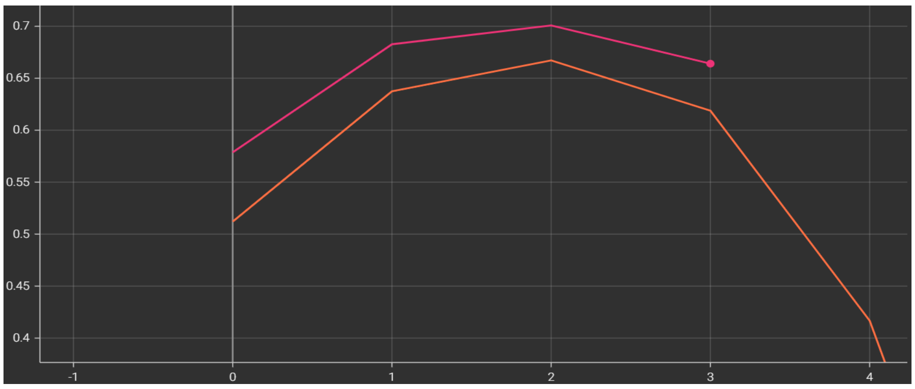
Results of experiments using LSTM layers.

Table 9: GRU model results

GRU layers	layer units	gradient clipping	epochs	minutes per epoch	F1-train	F1-test
2	1200	No	6	1166	0.0	0.0
2	1200	Yes	4	1431	0.664	0.670
2	400	No	8	476	0.780	**0.753**

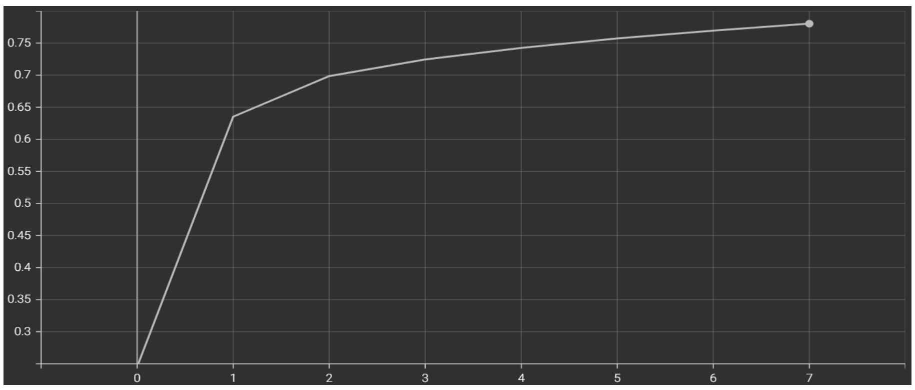
Results of experiments using GRU layers.

Figure 14: GRU unstable learning process graph



Graph showing the learning process of the GRU models from the first two rows of table 9 with(pink) and without(orange) gradient clipping. The vertical axis represents the training F1-score, and the horizontal axis represents the number of epochs minus 1.

Figure 15: GRU stable learning process graph



Graph showing the learning process of the GRU model in the third row of table 9. The learning rate for this model was lowered from 0.001 to 0.0001. The vertical axis represents the training F1-score, and the horizontal axis represents the number of epochs minus 1.

Table 10: Transformer model hyperparameter tuning

heads	dense units in transformer	epochs	minutes per epoch	F1-train	F1-test
1	10	2	882	0.708	0.718
1	50	2	683	0.702	0.714
3	10	2	1077	0.721	**0.730**

Results of tuning of the number of heads and number of dense units in the transformer model.

Table 11: Transformer model results

heads	dense units in transformer	epochs	minutes per epoch	F1-train	F1-test
3	10	6	953	0.765	**0.758**

Transformer model that includes a video classification module.

Table 12: DBoF model results

clusters	**two stream**	**gating**	**epochs**	**minutes per epoch**	**F1-train**	**F1-test**
2048	No	No	6	82	0.783	0.746
2048	Yes	No	6	99	0.790	0.751
2048	No	Yes	6	85	0.782	0.746
4096	Yes	No	6	171	0.792	0.751
4096	Yes	Yes	6	184	0.794	**0.753**
6500	Yes	No	6	367	0.789	0.750

Results of experiments using DBoF pooling layers.

Table 13: NetVLAD model results

clusters	**two stream**	**gating**	**epochs**	**minutes per epoch**	**F1-train**	**F1-test**
100	Yes	No	5	83	0.828	0.763
100	Yes	Yes	5	98	0.824	**0.767**
256	Yes	Yes	6	438	0.860	0.759

Results of experiments using NetVLAD pooling layers.

Table 14: NetFV model results

clusters	**two stream**	**gating**	**epochs**	**minutes per epoch**	**F1-train**	**F1-test**
60	Yes	No	6	111	0.817	0.760
60	Yes	Yes	7	121	0.832	**0.764**
128	Yes	Yes	6	247	0.828	0.761

Results of experiments using NetFV pooling layers.

Table 15: Best model results

gating	epochs	minutes per epoch	F1-train	F1-test	test precision	test recall
No	7	140	0.842	0.754	0.788	0.723
Yes	6	104	0.837	**0.778**	0.825	0.734

Results of combining the dense layer and dropout configuration from the last row of table 7, with the NetVLAD pooling layers in the third row of table 13, with and without context gating.

7 Discussion

There are many different ways of performing pooling on a tensor. This project explores four such methods and shows the results in tables 6,12,13 and 14. As mentioned, all learnable pooling models use the dense layer configuration in the third and fourth row of 6, which yields a test F1-score of **0.761**. Knowing this we can rank the pooling methods by their highest test F1-score where DBoF has **0.753** which is the lowest. The second best is the NetFV pooling with **0.764**, and the best test F1-score for pooling models was achieved by NetVLAD with **0.767**. It was expected that learnable pooling methods would make a more informative video representation than average pooling, and we observe that this is the case for learnable pooling models other than DBoF. This shows that a pooling method can be learned and used for better performance instead of a trivial method like average pooling.

For average pooling, the results of splitting the visual and audio features are shown in table 5. No noticeable improvement from the models in 4 were observed. For learnable pooling models, the results are shown in the first two rows of 12. Splitting the visual and audio features improved the test F1-score by 0.7%, so this model design was used in all other learnable pooling models.

The effect of context gating varies among the different models. In table 6, we observe that context gating improves the test F1-score in the range 0-0.3%, with the exception of the logistic model which experiences a 7% improvement, which is likely due to context gating adding complexity to the already underfitting logistic model. For the DBoF models in table 12, we observe an example of context gating decreasing the F1-score by 0.7% when comparing the second and third row, and an example of it increasing the F1-score by 0.3% when comparing the fourth and fifth row. For both the NetVLAD and NetFV models in tables 13 and 14, we observe a 0.5% increase in test F1-score in both cases when adding context gating. In general there was no large increase in computational time when adding context gating to a model, so based on the results where dropout was not used, context gating is a positive addition to average pooling and learnable pooling methods, with the exception of DBoF where the addition of context gating did not provide reliable improvements.

Table 7 shows the results of adding dropout to two average pooling models that yielded a high training F1-score in table 4. Firstly, dropout with a dropout rate of 0.25 was added to two architectures that also used context gating, yielding roughly a 1.3% improvement from the results in table 6. In the last 4 rows of table 6 it was observed that context gating had a nearly regularizing effect in the sense that it lowered the training F1-score, so because of this, another experiment was done where the context gating was removed and the dropout rate was increased to 0.35. This resulted in an even better improvement of around 1.8%. This experiment resulted in the best video-level model that would be combined with the best learnable pooling method to create the model with best performance.

In this project the attempts at using RNNs yield the lowest performances. The result of the first attempt at training a GRU model, shown in the first row of table 9, yielded an unstable learning process where the F1-score started decreasing after the third epoch as seen in 14. This was suspected to be caused by exploding gradients so gradient clipping was tested in the second row. Although adding gradient clipping did increase the training F1-score, it did not solve the problem of decreasing performance. It was then speculated that the choice of architecture and a too high learning rate caused the model to converge quickly to a suboptimal solution, so for a third attempt, the learning rate and the number of units in the GRU layers were lowered. This configuration did not encounter the same problem and the result of this is shown in the third row of table 9, and the learning process can be seen in figure 15. We observe that the less complex GRU model yields a test F1-score of **0.753**, which is higher than that of the best LSTM model which was **0.746**.

Inspired by their performance in the field of natural language processing, transformer models were used in this project. We observe in tables 8,9,11 that the transformer models yield the best performance among the sequential models, which is a test F1-score of **0.758**, even though little parameter tuning was done in this project. The hyperparameter tuning for the transformer models is shown in table 10. It can be seen in the second row that increasing the number of dense units did not improve the performance. The third row shows that increasing the number of self attention heads significantly improves performance. With this little tuning, it is very likely that the model's performance can be improved upon. Based on the results, the transformer encoder is a viable method for video classification tasks.

Using what was learned in the mentioned results, the best model was designed by combining the NetVLAD pooling method from the second row of table 13 with the dense and dropout layer configuration from the third and fourth row of table 7. The results of this are shown in table 15, where a test F1-score of **0.778** is achieved. This is the highest test F1-score in this project using a learnable pooling method, but unexpectedly, it is only slightly higher then the best score for average pooling, showing that, although simple, average pooling can provide an informative representation of the frames in a video. In table 15, the test precision and test recall are included. We observe that the precision is significantly higher than the recall. A high precision means that the labels that the model predicts are often the true labels while a low recall means that often not all the true labels are among the models predicted labels. This was expected because as a result of the dataset being imbalanced, some labels appear in very few videos which makes it hard for a model to learn to classify them.

Although the project explored different deep learning models and different methods of improving their performance, more time could have been spent on systematic hyperparameter tuning, especially for sequence models. It is likely possible to increase the individual model performances by further hyperparameter tuning, and adjustments to the network architectures.

Tables 4 and 6 show the results of similar average pooling models applied to different amounts of data. From those tables, on average, increasing the data amount from 30% to 100% yields roughly a 1.8% increase in test F1-score. This is a considerable improvement in performance compared to other attempts at improving the models, and highlights that using more data is a powerful tool in deep learning. Based on these results, it is reasonable to believe that the use of the whole dataset would give a larger performance boost to all models than slight changes to architectures or hyperparameters. There is also the possibility that the performance of the sequential and learnable pooling models scales differently with more data. For example, the performance of sequential models in this project is worse that pooling models, but that could have been different if the whole frame-level dataset was used. Based on this, a desirable way of improving model performance in future works would be to utilize more of the frame-level dataset.

The tables in the result section include the average number of minutes spent on training one epoch, as it would be interesting to include the training time in the considerations when comparing the performances of different models. Unfortunately these results have a high margin of error. This is because the computations were ran on a server that distributes computational resources equally among students that are using it. Consequently, the computing speed at a time would depend on how many students are using the server simultaneously. An example of this can be seen in table 10 when comparing the minutes per epoch in the first two rows. The second row has a higher complexity in the form of having more units in the dense layers of the transformer encoder, so it is expected to have a longer training time, but instead the epochs in the second row were computed 23% faster. The same thing happens when comparing the third row of 10 and the result in 11. The model in 11 has two additional dense layers to train between the transformer encoder and the classification layer, but on average computes each epoch 12% faster. In general the results show that the slowest models were the sequential models, followed by learnable pooling models, and the fastest were the average pooling models, which was expected.

8 Conclusion

The objective of this project was to to implement and experiment with different types of deep learning models that can learn from the Youtube-8M dataset, and compare their performances. This was done by implementing RNNs and Transformer models which are variants of sequential methods, and Average pooling, DBoF, NetVLAD, and NetFV which are variants of pooling methods. Experiments with hyperparameter tuning, network architecture, regularization and context gating were done to explore ways of improving individual model performance. Among sequential models, the newer Transformer model outperformed more classical temporal approaches like the LSTM and GRU, although little hyperparameter tuning was performed on these models due to the lengthy computational times. In this project, with the exception of DBoF models, pooling based models are observed to have better performance than the best sequential model, and among these NetVLAD reached the highest testing F1-score of **0.778**. Dropout proved to be a powerful tool at increasing the performance of overfitting models, and in general, context gating showed to have a positive effect on performance of pooling models, with the exception when it was combined with DBoF models and dropout layers.

It is worth noting that the frame-level models were implemented on a small subset of an highly imbalanced dataset. This resulted in many labels having less that a hundred samples to learn from, making it unlikely for the models trained in this project to classify them in multi-label video classification. Because of this, although it should be possible to increase the individual model performances by more systematic hyperparameter tuning, and adjustments to the network architectures, it is reasonable to believe that the most significant improvement would likely be attained if the whole frame-level dataset was utilized, which is left for future work. In addition to this, future work should also focus on exploring ways to ensemble the best models into one for optimal performance.

References

[1] Ivan Laptev Antoine Miech and Josef Sivic. *Learnable pooling with Context Gating for video classification.* URL: https://arxiv.org/pdf/1706.06905.pdf (visited on 23rd May 2022).

[2] Dan Becker. *Rectified Linear Units (ReLU) in Deep Learning.* URL: https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook (visited on 28th May 2022).

[3] ALEKSEY BILOGUR. *Keras optimizers.* URL: https://www.kaggle.com/residentmario/keras-optimizers/notebook#Adam (visited on 19th Dec. 2021).

[4] Jason Brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks.* URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (visited on 9th June 2022).

[5] Yash BohraJason Brownlee. *The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks.* URL: https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/ (visited on 7th June 2022).

[6] Ram Cherukuri. *What Is int8 Quantization and Why Is It Popular for Deep Neural Networks?* URL: https://se.mathworks.com/company/newsletters/articles/what-is-int8-quantization-and-why-is-it-popular-for-deep-neural-networks.html (visited on 21st May 2022).

[7] Francois Chollet. *A ten-minute introduction to sequence-to-sequence learning in Keras.* URL: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (visited on 28th May 2022).

[8] Francois Chollet. *Deep Learning with Python.* Manning Publications, 2021.

[9] Google Cloud. *Google Cloud YouTube-8M Video Understanding Challenge.* URL: https://www.kaggle.com/competitions/youtube8m/overview (visited on 23rd May 2022).

[10] Ji Wu He-Da Wang Teng Zhang. *The Monkeytyping Solution to the YouTube-8M Video Understanding Challenge.* URL: https://arxiv.org/pdf/1706.05150.pdf (visited on 23rd May 2022).

[11] Jimmy Lei Ba Diederik P. Kingma. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.* URL: https://arxiv.org/pdf/1412.6980v9.pdf (visited on 23rd May 2022).

[12] Keith L. Downing. *Backpropagation: The Good, the Bad and the Ugly.* URL: https://www.idi.ntnu.no/emner/it3030/lectures/deep-lecture-2.pdf (visited on 10th May 2022).

[13] Keith L. Downing. *Recurrent Neural Networks (RNNs).* URL: https://www.idi.ntnu.no/emner/it3030/lectures/deep-lecture-5.pdf (visited on 16th May 2022).

[14] Keith L. Downing. *Regularization and Optimization of Backpropagation.* URL: https://www.idi.ntnu.no/emner/it3030/lectures/deep-lecture-3.pdf (visited on 5th June 2022).

[15] Jorge S´anchez Florent Perronnin and Thomas Mensink. *Improving the Fisher Kernel for Large-Scale Image Classification.* URL: https://lear.inrialpes.fr/pubs/2010/PSM10/PSM10_0766.pdf (visited on 15th May 2022).

[16] Google. *Updated Dataset.* URL: https://research.google.com/youtube8m/download.html (visited on 10th May 2022).

[17] Per Kristian Hove. *Markov.* URL: https://wiki.math.ntnu.no/drift/stud/ommarkov (visited on 23rd Dec. 2021).

[18] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[19] Murat Karakaya. *How to solve Classification Problems in Deep Learning with Tensorflow Keras?* URL: https://medium.com/deep-learning-with-keras/how-to-solve-classification-problems-in-deep-learning-with-tensorflow-keras-6e39c5b09501 (visited on 12th May 2022).

[20] Murat Karakaya. *How to solve Multi-Label Classification Problems in Deep Learning with Tensorflow Keras?* URL: https://medium.com/deep-learning-with-keras/how-to-solve-multi-label-classification-problems-in-deep-learning-with-tensorflow-keras-7fb933243595 (visited on 21st May 2022).

[21] Andrea Vedaldi Karen Simonyan and Andrew Zisserman. *Deep Fisher Networks for Large-Scale Image Classification.* URL: https://www.robots.ox.ac.uk/~vgg/publications/2013/Simonyan13b/simonyan13b.pdf (visited on 15th May 2022).

[22] Pavel Ostyakov Elizaveta Logacheva Roman Suvorov Vladimir Aliev Gleb Sterkin Oleg Khomenko and Sergey I. Nikolenko. *Label Denoising with Large Ensembles ofHeterogeneous Neural Networks.* URL: https://static.googleusercontent.com/media/research.google.com/en//youtube8m/workshop2018/c_07.pdf (visited on 23rd May 2022).

[23] Joos Korstanje. *The F1 score.* URL: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6 (visited on 18th Dec. 2021).

[24] Ajitesh Kumar. *Gaussian Mixture Models: What are they when to use?* URL: https://vitalflux.com/gaussian-mixture-models-what-are-they-when-to-use/ (visited on 15th May 2022).

[25] VIJAYSINH LENDAVE. *LSTM Vs GRU in Recurrent Neural Network: A Comparative Study.* URL: https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/ (visited on 16th May 2022).

[26] Vinayak Mishra. *Bag Of Visual Words.* URL: https://medium.com/analytics-vidhya/bag-of-visual-words-bag-of-features-9a2f7aec7866 (visited on 12th May 2022).

[27] Relja Arandjelovic Petr Gronat Akihiko Tori Tomas Pajdla and Josef Sivic. *NetVLAD: CNN architecture for weakly supervised place recognition.* URL: https://arxiv.org/pdf/1511.07247.pdf (visited on 10th May 2022).

[28] Sayak Paul. *Video Classification with Transformers.* URL: https://keras.io/examples/vision/video_transformers/ (visited on 28th May 2022).

[29] Florent Perronnin and Christopher Dance. *Fisher Kernels on Visual Vocabularies for Image Categorization.* URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.7388&rep=rep1&type=pdf (visited on 15th May 2022).

[30] Ashish Vaswani Noam Shazeer Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin. *Attention Is All You Need.* URL: https://arxiv.org/abs/1706.03762 (visited on 28th May 2022).

[31] Google Research. *The 2nd YouTube-8M Video Understanding Challenge.* URL: https://www.kaggle.com/c/youtube8m-2018 (visited on 23rd May 2022).

[32] Google Research. *The 3rd YouTube-8M Video Understanding Challenge.* URL: https://www.kaggle.com/c/youtube8m-2019 (visited on 23rd May 2022).

[33] Google Research. *YouTube-8M: A Large-Scale Video Classification Benchmark.* URL: https://arxiv.org/pdf/1609.08675.pdf (visited on 10th May 2022).

[34] Herve Jegou Matthijs Douze Cordelia Schmid and Patrick Perez. *Aggregating local descriptors into a compact image representation.* URL: https://lear.inrialpes.fr/pubs/2010/JDSP10/jegou_compactimagerepresentation.pdf (visited on 10th May 2022).

[35] Miha Skalic and David Austin. *Building a Size Constrained Predictive Model for Video Classification.* URL: https://static.googleusercontent.com/media/research.google.com/en//youtube8m/workshop2018/c_14.pdf (visited on 23rd May 2022).

[36] Ava Soleimany. *MIT 6.S191: Recurrent Neural Networks and Transformers.* URL: https://www.youtube.com/watch?v=QvkQ1B3FBqA&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=2&ab_channel=AlexanderAmini (visited on 15th June 2022).

[37] NTNU Stefanie Muff Department of Mathematical Sciences. *Module 11: Neural Networks.* URL: https://htmlpreview.github.io/?https://github.com/stefaniemuff/statlearning/blob/master/11NNet/11Nnet.html (visited on 18th Dec. 2021).

[38] NTNU Stefanie Muff Department of Mathematical Sciences. *Module 3: Linear Regression.* URL: https://htmlpreview.github.io/?https://github.com/stefaniemuff/statlearning/blob/master/3LinReg/3LinReg.html (visited on 19th Dec. 2021).

[39] Google Brain Team. *Masking and padding with Keras.* URL: https://www.tensorflow.org/guide/keras/masking_and_padding (visited on 21st May 2022).

[40] Google Brain Team. *TensorBoard: TensorFlow's visualization toolkit.* URL: https://www.tensorflow.org/tensorboard (visited on 23rd May 2022).

[41] Google Brain Team. *tf.data.Dataset*. URL: https://www.tensorflow.org/api_docs/python/tf/data/Dataset (visited on 21st May 2022).

[42] Google Brain Team. *tf.keras.optimizers.schedules.CosineDecayRestarts*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/CosineDecayRestarts (visited on 3rd June 2022).

[43] Google Brain Team. *The Functional API*. URL: https://www.tensorflow.org/guide/keras/functional (visited on 23rd Dec. 2021).

[44] Great Learning Team. *An Introduction to Bag of Words (BoW) — What is Bag of Words?* URL: https://www.mygreatlearning.com/blog/bag-of-words/ (visited on 12th May 2022).

[45] Tensorflow. *TFRecord and tf.train.Example*. URL: https://www.tensorflow.org/tutorials/load_data/tfrecord (visited on 10th May 2022).

[46] Eric W Weisstein. $L^2 - Norm$. URL: https://mathworld.wolfram.com/L2-Norm.html (visited on 7th June 2022).

[47] Wanshun Wong. *What is Gradient Clipping?* URL: https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48 (visited on 9th June 2022).

[48] Thomas Wood. *What is the Softmax Function?* URL: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (visited on 28th May 2022).

[49] Tsun-Yi Yang. $LOUPE_Keras$. URL: https://github.com/shamangary/LOUPE_Keras (visited on 15th May 2022).