FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Towards a dataset for estimation of keyboard fingerings
Student:	Filip Danielsson
Supervisor:	doc. RNDr. Pavel Pecina, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Automatic estimation of keyboard fingerings is a process that could be useful as a tool in music learning environments for both beginners and intermediates.

The goal of the thesis is to implement a data processing pipeline that can be used to generate a dataset for training predictive models for estimation of keyboard fingerings by exploiting publicly available video recordings of playing the piano (e.g. from Youtube) and existing models for piano music transcription and hand segmentation (e.g. https://arxiv.org/pdf/2010.01815.pdf).

The work will include the following steps:
1. Survey state of the art methods for hand segmentation and piano transcription
2. Preprocess input videos using computer vision techniques, automatically locate the keyboard and segment the keys.
3. Generate note and hand position data using existing state of the art methods for hand segmentation and piano transcription
4. Identify which fingers correspond to the generated note onsets.
5. Experiment with data correction using heuristics between the note and hand position data.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 3 February 2023 in Prague.

Bachelor's thesis

TOWARDS A DATASET FOR ESTIMATION OF KEYBOARD FINGERINGS

Filip Danielsson

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: doc. RNDr. Pavel Pecina, Ph.D.
May 16, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Danielsson Filip. *Towards a dataset for estimation of keyboard fingerings.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

I express gratitude to my supervisor, doc. RNDr. Pavel Pecina, Ph.D., for their invaluable guidance during the course of this thesis. Additionally, a heartfelt thanks to my friends and family for their constant support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 16, 2024

Abstract

In this work, a system was developed that combines keyboard detection, key segmentation, piano transcription, and hand-tracking into a pipeline for general piano performance videos recorded from an overhead perspective. A model was trained for localizing keyboards, and the background of the keyboard scenes was estimated, followed by a procedure for segmenting and labeling the keys. Keyboard detection, segmentation, and video sectioning were tested on the 100 latest videos from 5 prospective YouTube channels. Furthermore, the content of these videos was analyzed for data points relevant to pianist movements. Keyboard fingerings were predicted using a distance metric between the hands and key bounds, achieving an 82% accuracy. Utilizing the knowledge of hand positions also demonstrated up to a 6.6% improvement in piano transcription F1 score. An evaluation video containing 1227 notes with manually labeled fingerings and ground truth midi was created to derive the aforementioned fingering and transcription results.

Keywords piano fingering, dataset, data correction, piano transcription, hand tracking

Abstrakt

V této práci byl vyvinut postup, který kombinuje detekci klaviatury, segmentaci kláves, klavírní transkripci a odhad pozice rukou za účelem odhadu prstokladu z klavírních videí nahrávaných shora. Byl natrénován model pro lokalizaci klaviatury, následovaný algoritmem pro segmentaci a označování kláves. Lokalizace a segmentace klaviatury spolu s dělením videí na sekce byly testovány na 100 nejnovějších videích z pěti vybraných YouTube kanálů. Dále byl analyzován obsah těchto videí z hlediska datových bodů relevantních pro hru na klavír. Prstoklad byl odhadnut pomocí metriky vzdálenosti mezi rukama a hranicemi kláves, čímž bylo dosaženo přesnosti až 82 %. Využití znalostí o pozicích rukou také vedlo ke zlepšení F1 skóre klavírní transkripce o až 6,6 %. Výše zmíněné výsledky byly odvozeny z videa se 1227 notami s referenčním MIDI a ručně označenými prstoklady.

Klíčová slova klavírní prstoklad, datová sada, korekce dat, klavírní transkripce, odhad pozice rukou

Abbreviations

AMT	Automatic Music Transcription
DFT	Discrete Fourier Transform
DCT	Discrete Cosine Transform
GPU	Graphical Processing Unit
HSL	Hue Saturation Lightness
IOU	Intersection Over Union
MFCC	Mel Frequency Cepstral Coefficients
MIDI	Musical Instrument Digital Interface
TET	Twelve-Tone Equal Temperament
YOLO	You Only Look Once

Chapter 1
Introduction

Learning the piano and hand movements associated with it is a difficult multivariate task. Learners of all levels could benefit from seeking clarity on their approach from a teacher or an automatic system. Choosing appropriate keyboard fingerings is one technical aspect that could be enhanced with guidance, especially for beginners. Videos of piano performances contain valuable information about keyboard fingerings, movement, and hand shape that could be utilized in learning systems. A pipeline for extracting information about the notes, keys, and hands from readily available videos on a large scale would allow for quantitative research on pianist mechanics. The accuracy of state-of-the-art piano transcription and hand-tracking models has reached a point where such features could be produced automatically from generally available video recordings. Previous works on videos are from a single source and require ground truth MIDI files hence limiting the use cases.

Researchers have tried modeling keyboard fingerings in various ways using search techniques, cost functions, Markov models, neural networks, and others. Having a dataset for optimizing and testing these algorithms is crucial. The PIG dataset [1] is as of this moment, the largest hand-labeled dataset for piano fingering, containing parts from 150 annotated pieces from various Western classical composers. Choosing the correct finger pattern is a complex process that can have varying solutions depending on attributes such as hand size, tempo, personal preference, and skill level. It has been shown that human fingering agreement is 71.4% on a subset of the PIG dataset[1]. Using just labeled notes for training fails to take the more complex reasons behind the finger choices into account and will likely never surpass the human agreement.

Video recordings of people playing the piano have, in some ways, been studied before [2, 3]. High-quality overhead recordings contain enough information to determine which finger was used and have additional information about the size, shape, and movement of the hands. Recording an overhead video is less labor-intensive than manually annotating a score, and many videos already exist in a suitable format on online platforms such as YouTube. Models for transcribing piano audio into notes, such as [4], are now at a point where they could be used in conjunction with videos that do not have the corresponding transcription.

As of now, no system has been found that is capable of automatically extracting and processing general overhead piano videos with varying settings and performers. In order to work with different sources, it would have to do the following automatically:

1. Detect the presence of piano music and transcribe the played note pitches and timings
2. Detect in which frames the keyboard is located and its position within each frame.
3. Segment the keys and label the pitches despite obstructions and varying shifts.
4. Detect the hand presence, location, and handedness, and estimate the pose of the fingers.
5. Use the above to estimate the fingering used.

Such a pipeline could be used on many existing video sources on the internet to generate vast amounts of data on the keys, notes, hands, and estimated fingerings. The joint data could be used to train models for the estimation of fingerings. The individual movements and preferences between performers could be quantified and compared, possibly surpassing the human agreement baseline for fingering given additional information about the person.

This work is a first step towards automatically utilizing more generally available video recordings for modeling and analyzing the mechanics of piano performances. Achieving the desired system will require solving multiple problems at each step of the pipeline. The text of this thesis first studies and reviews the prerequisites and techniques of each significant part. Followed by a section on the implementation and the specific choices made in applying the described techniques to this use case. In the end, the results of each part are evaluated and discussed.

1.1 Previous Works

One previous work was identified that has worked with extracting data from online piano videos [3]. The authors created a dataset of piano fingerings from videos with a similar format and attempted to model these fingerings using the dataset. The work relied on video creators providing ground truth MIDI files with the videos. The amount of available videos with corresponding MIDI files is much smaller, and manual work is needed to download and pair each MIDI file with the video. Finding videos with ground truth MIDI files in good lighting conditions for specific pieces might not be possible. CycleGAN [5] was thus used to fine-tune a hand pose estimator in bad light conditions. Additionally, the authors had to synchronize the MIDI files with the videos by using a heuristic and maximizing fingering confidence scores.

The data processing pipeline relies on hand-picked videos in the correct format. For example, heuristics such as estimating the keyboard as the largest continuous bright area will not work in cases where a bright musical score is also present. Such an approach can not be used for the automatic identification of relevant videos and sections. The key boundary estimation and labeling process is not described in the paper but can affect the fingering prediction performance. Furthermore, the system relies on having the full 88 white key range visible, avoiding the need to estimate key labels from hand positions in a zoomed video [3].

In this thesis, note onsets were estimated using a high-resolution transcription system from the audio. MIDI estimates are based on the audio directly so synchronization will not be needed. The downside of automatic transcription, lowered transcription accuracy, is in part remedied by using the additional video input. Additionally, a more robust processing pipeline was developed, allowing for a fully automatic system for detecting and extracting relevant sections on a much broader range of videos.

Chapter 2
Background

2.1 Neural networks

2.1.1 Deep Feedforwards Networks

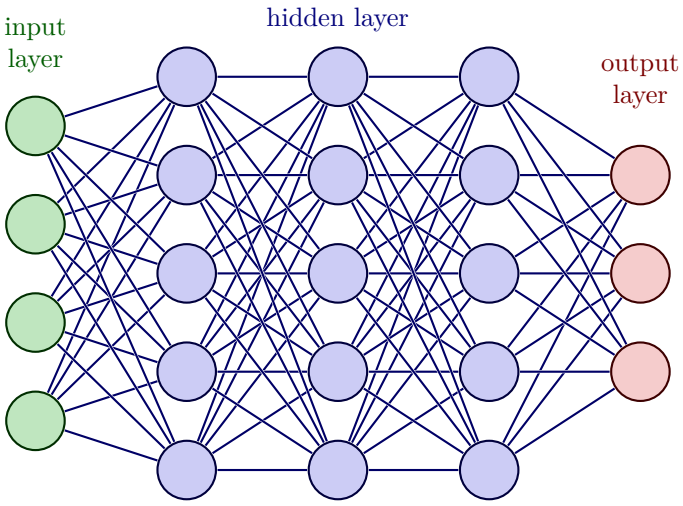

Figure 2.1 A neural network with 3 hidden layers [6]

A fully connected feed-forward neural network can be seen as multiple layers of artificial neurons. A fully connected network will have at least the output layer with any number of additional hidden layers in between. It is fully connected because each neuron is connected to every neuron in the previous layer. Each neuron has a set of numbers called weights, one for every neuron in the previous layer. The output of a single neuron is produced by multiplying the previous layer with these weights, adding a bias, and applying an activation function. This computation can be described using vector notation where the weights are a matrix and the outputs of the layer are vectors. The vector output of layer n $\mathbf{h_n}$ can be described as:

$$\mathbf{h_n} = \sigma_n(\mathbf{W_n^T h_{n-1}} + \mathbf{b_n}) \tag{2.1}$$

where $\mathbf{W_n}$ is the matrix of weights, $\mathbf{b_n}$ is the vector of biases and σ_n is the activation function of layer n. h_0 is the input layer $\mathbf{x}$.

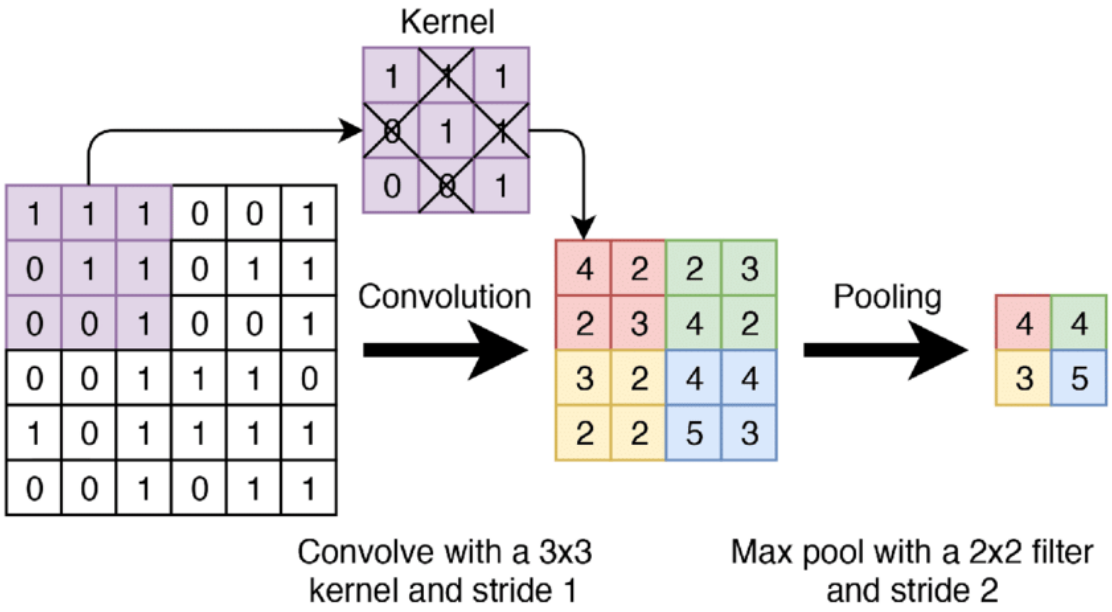

Figure 2.2 A sample convolutional and max pooling layer [8]

The weights and the bias are ”learnable” and will be estimated using an automatic algorithm called gradient descent. The activation functions σ have to be chosen, common activation functions are relu, sigmoid, or htan. Activation functions should be non-linear so that non-linear relationships between $\mathbf{x}$ and $\mathbf{y}$ can be estimated but also mostly differentiable [7].

2.1.2 Convolutional Neural Networks

Convolutional layers are created using the convolution operation, where a kernel of a set size containing weights is convolved over the input before the bias and activation are applied. They are largely invariant to the position of the input and can hence pick up features regardless of their location. Multiple convolutional layers stacked after each other will build hierarchies of features. Some common properties are the stride, size, and number of kernels. The output size will be proportional to the mentioned properties, as well as the input size.

A 1D discrete convolution is defined as follows:

$$s(t) = (x * w)(t) = \sum_{n=1}^{N} x(n)w(t-n) \tag{2.2}$$

Convolutions can work in higher dimensions as well. Image input would commonly be used together with 2D convolutions:

$$S(i,j) = (x * w)(i,j) = \sum_{m}\sum_{n} x(m,n)w(i-m, j-n) \tag{2.3}$$

where $*$ is the convolutional operation, x is the input an w is the kernel containing weights.

Pooling layers share properties similar to those of the convolutional layer but without any learnable parameters. They reduce the size of the previous layer by pooling together adjacent inputs into one. This could be done by averaging *average pooling* or taking the maximum *max pooling*. Pooling layers share similar properties, such as stride and kernel size [7].

2.1.3 Training

The process of finding the learnable parameters of a network (weights, biases, kernels...) is called training. Neural networks are generally trained using some variation of the gradient descent algorithm.

Given a set of training data x_i and y_i, a network would be considered to have the optimal weights θ if some loss function $\mathcal{L}$ is minimal over that data. The goal of the loss function is to quantify the severity of the prediction error in a stable way, given the characteristics of the output.

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i; \theta)) \tag{2.4}$$

Backpropagation enables estimation of the minimum of $L(\theta)$ by repeatedly taking steps in the opposite direction of the gradient. After initializing θ, the algorithm is usually divided into two steps, the forward and backward pass. In the forward pass, all intermediate layers from the input are computed and stored for each element of the training set. In the backward pass, the gradient of $L(\theta)$ is calculated layer by layer using the chain rule and derivatives of intermediate operations.

$$\nabla_\theta L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(y_i, f(x_i; \theta)) \tag{2.5}$$

where $\mathcal{L}$ is the loss function, y are the expected outputs, and $f(x_i; \theta)$ is the neural network output given the input x_i and weights θ.

θ is then updated by $-\nabla_\theta L(\theta)\eta$, where η is the learning rate or step size. The exact update procedure is generally more complex and governed by an optimizer algorithm [7].

Training is usually done on smaller subsets of the training data called batches. Larger batches are more memory intensive, require less training time, and can lead to lowered performance. The gradient descent process can be used on different layers and architectures as long as each weight-dependent operation in the chain between the input and output is differentiable. Modern deep learning frameworks allow users to define networks using nested high-level functions that make up a computational graph. Activation functions, the loss function, the optimizer, layer types, and their properties are generally considered to be hyperparameters. They greatly affect the performance and accuracy of the network, but their optimization is usually handled outside of the gradient descent process.

2.2 Computer Vision

2.2.1 Images

The field of Computer Vision (CV) models perception from visual inputs. Inputs of computer vision algorithms can be built for individual images or video sequences. They can work with direct captures of the real world, such as photos and scans, or from a device like digital artwork, 3D renderings, or vector graphics. Raw rasterized images are most commonly represented in red, green, and blue (RGB) format, and different proportions of these 3 colors can create any color in the visible spectrum. Changing the color space as a preprocessing step might represent the image in a format more suitable to the task at hand. Examples of commonly used color spaces are grayscale or hue, saturation, value (HSV) [9]. Image format, resolution, frame rate, compression, and other input qualities should be taken into account in the computer vision design process.

2.2.2 Template Matching

Template matching is a basic computer vision algorithm for finding the occurrences of a template image inside a source image. The algorithm iterates over each segment of the source image and computes a similarity metric between that segment and the template. Locations where the similarity metrics exceed a threshold or are maximal can be considered a match. The algorithm and the similarity metrics are usually used on one-dimensional grayscale images, but it could adapted to colors if they are significant. Similarity to the template can be calculated as the square difference or the normalized cross-correlation (NCC). The algorithm can be optimized by computing the similarities in the frequency domain with the Fast Fourier Transform. Template matching does not extract any complex features from the template and only works well when close to exact matches need to be found; changes in scale and rotation will also change the result. It might be most useful on screen-captured or generated images showing a set number of graphics in standard sizes [10].

2.2.3 Background Estimation

Estimating the background from a still camera sequence of images is closely related to extracting the foreground from an image and detecting change or movement. These methods can be used on footage from roads and other spaces to count, track, and model the behavior of people, vehicles, and other objects. SBMNet [11] is a dataset and competition for finding and comparing background estimation models on a variety of footage targeting common issues such as camera motion or changes in lighting.

The simplest methods for background estimation are the pixel-wise mean, median, and histogram. The mean of a pixel across all frames of a video will diminish the less common occurrences. Artifacts of moving objects will be visible if the number of frames is low or if the objects are slow. The median will yield good results on a pixel if the background is present in more than 50% of the frames. On the other hand, computing the median can also be computationally demanding because all the frames have to be held in memory at once. Other techniques have used Principal Component Analysis (PCA) on the pixels in time or neural networks [12].

2.2.4 Object Detection

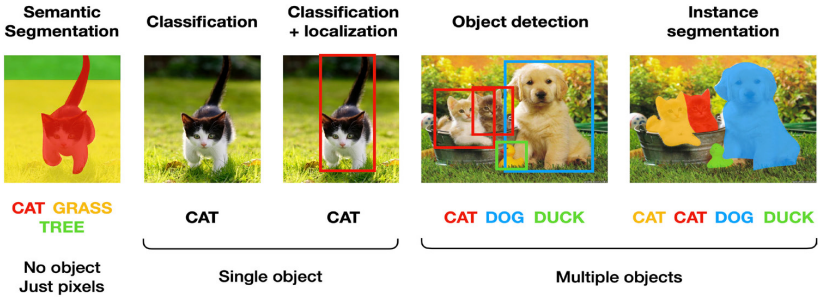

Figure 2.3 Comparison of computer vision tasks [13]

Object detection is the task of locating and classifying multiple objects in an image. Locations of objects are specified with bounding boxes that surround the whole object. Unlike segmentation, the bounding boxes may contain parts of other objects and backgrounds. Object detection systems may also predict a confidence score of an object being present inside the bounding box. The resulting classification of each box can be one class or a probability distribution over all possible classes. When multiple objects are located within the same bounding box, a distribution might better represent the real class of that region. Systems and datasets should be adapted to find a balance between common properties such as classification and localization accuracy, latency, object and class count, object scale/zoom, image quality, style, etc. [14].

Researchers often use public datasets with common objects such as ImageNet, Coco, or PascalVOC to train and benchmark their models. Each dataset might have subsets for different tasks with varying class sizes and image resolutions alongside leaderboards to aggregate results. An object detection model performs well if its bounding box predictions are close to the ground truth and correctly assigns the class to each one. It should also be evident when the model produces false positives, for example, by classifying the whole input image as all classes. The Intersection Over Union (IOU) is used to quantify the similarity of two bounding boxes [14].

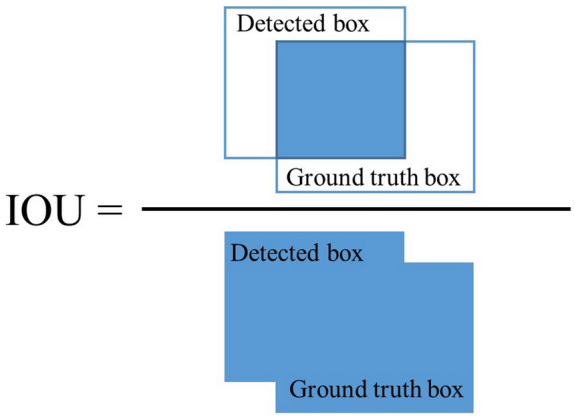

Figure 2.4 Figure depicting the intersection over union [15]

In the context of object detection, a predicted bounding box is considered a True Positive (TP) if it has an IOU over a certain threshold with any of the ground truth bounding boxes of the same class and is considered to be a False Positive (FP) if it is under the same threshold for all ground truth bounding boxes. All ground truth bounding boxes that didn't match any of the predicted ones are False Negatives (FN). Given an IOU threshold, standard Precision and

Recall metrics can be determined:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.7}$$

where:

$$\text{TP} = \text{True Positives, FP} = \text{False Positives, FN} = \text{False Negatives.}$$

Precision and recall measure the ability to exclude False Positives and False Negatives, respectively. The joint error can be measured using the F1 score:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.8}$$

Precision, Recall, and the F1-score are all measured between 0 and 1, where bigger is better. Models may produce redundant bounding box predictions with low confidence scores. They can be removed by choosing another threshold under which predictions will not be included in the evaluation. Increasing the confidence threshold will exclude more bounding boxes and eventually increase the number of False Negatives. Increasing the IOU threshold will mark fewer bounding boxes as true positives, after which the number of False Positives increases. Finding the optimal balance of these thresholds will depend on the sensitivity to the different error types on a given task. To compare the overall performances across a range of thresholds Average Precision (AP) and Mean Average Precision (mAP) can be used. First, the Precision-Recall curve for each class is computed by gradually increasing the confidence threshold. Average Precision is calculated by averaging a number of evenly spaced Precision scores on the precision-recall curve. Mean average precision is the mean of AP across all classes [16].

Additionally, some models may produce predictions that are close to duplicate for the same region. The non-maximum suppression algorithm keeps the highest confidence bounding box in each region by deleting other overlapping bounding boxes of a lower confidence score [17].

Algorithm 1 Non-Maximum Suppression

Input: List of bounding boxes $\mathcal{B} = \{B_1, B_2, ..., B_n\}$, threshold t
Output: List of selected bounding boxes after NMS $\mathcal{B}_{\text{selected}}$
Initialize an empty list $\mathcal{B}_{\text{selected}}$
while $|\mathcal{B}| \neq 0$ **do**
  $B_i \leftarrow$ bounding box from $\mathcal{B}$ with highest confidence score
  Add B_i to $\mathcal{B}_{\text{selected}}$
  for B_j in $\mathcal{B}$ **do**
    if IoU$(B_i, B_j) > t$ **then**
      Remove B_j from $\mathcal{B}$
    end if
  end for
end while
return $\mathcal{B}_{\text{selected}}$

2.2.5 YOLO

You Only Look Once (YOLO) [18] is an approach to object detection with real-time capabilities. Object detection pipelines at the time of the release of YOLO were often divided into multiple steps (bounding box proposition, classification, refinement...). This approach was both difficult to optimize and slow. YOLO uses a convolutional neural network to predict both class probabilities and bounding boxes in one evaluation of the network. It is able to accomplish this by splitting the input image into a SxS-sized grid of cells and predicting a fixed number of confidence scores and bounding boxes relative to the positions of the cells, as well as a class probability for each cell. Grid cells that contain the center of an object are responsible for the bounding box prediction of that object. Any object intersecting a cell will contribute to its class probabilities. The confidence score is the IOU between the ground truth and the bounding box prediction. Predicted bounding boxes can be outside of each cell as long as the object center is inside the cell. The result can then be filtered using non-maximum suppression to reduce overlapping bounding boxes [18].

The downside of the initial approach was spatial constraints on objects. Each cell only predicted two boxes and one class, which could cause issues with groups of smaller objects such as flocks of birds. The YOLO architecture has been refined several times after the initial release. One of the main changes since has been using clustering techniques on the bounding boxes of the training dataset to initiate sizes and aspect ratios of prior/anchor bounding boxes and constrain the localization of the bounding boxes. Instead of predicting the exact bound relative to the cell the changes to the anchors are predicted [19].

2.3 Piano Transcription

2.3.1 Audio Processing

An audio waveform is a one-dimensional sequence of numbers reflecting the changes in air pressure around a microphone. The basic limiting characteristics of a captured signal are the maximum frequency and dynamic range. These are dependent on the sampling rate and bit depth of the microphone or storage format. Most high-definition recordings are captured at a sampling rate of at least 44,1kHz with a bit depth of 16. The upper limit of human frequency perception is approximately 20 kHz which is within the Nyquist frequency of 44,1 kHz. Various audio compression algorithms, such as MP3, can be applied to reduce the storage size and transmission speeds. Most modern headphones, speakers, and storage systems allow for stereo recordings where multiple inputs are compiled into a left and a right audio track, providing for a more immersive listening experience.

Music is a common use case for audio. Music usually follows certain structures that enable physical and digital representations other than waveforms. Many types of music, especially solo piano music, can be reduced to notes in time with corresponding pitches, durations, and strengths/velocities. Various forms of sheet music exist across cultures and time periods, some of which have digital counterparts. The MIDI file format contains information about musical events, such as notes being turned on or off with different velocities and instruments. Notes of an instrument in time can be represented formally as a 2-dimensional Time × Pitch array, often called a piano roll.

A tuning system determines which frequencies/pitches can be assigned to a note. Twelve-tone equal temperament (TET) is a prominent tuning system that divides the doubling of a frequency into 12 tones, each of which is spaced evenly on a logarithmic scale. The frequency of a note n relative to the base note A4=440Hz in twelve-tone equal temperament tuning is $440 * 2^{n/12}$Hz. The waveform of played notes is not only determined by frequency and amplitude but also by the specific timbre of the instrument and noise from surroundings or equipment. The same pitch performed by two different instruments or in different environments might produce vastly different waveforms, but there will be similar periodicity at the common pitch. Musical transcription involves converting these waveforms with music into structured musical formats such as sheet music or MIDI.

2.3.1.1 Fourier Transform

Transcribing music requires extracting information about frequency representation at each point in time. The Fourier Transform (FT) [20] transforms continuous complex functions $x(t)$ into the frequency domain by integrating the function at each frequency f around the complex unit circle:

$$X(f) = \int_{-\infty}^{\infty} x(t)(\cos(2\pi ft) + j\sin(2\pi ft))\,dt = \int_{-\infty}^{\infty} x(t) \cdot e^{-j2\pi ft}\,dt \tag{2.9}$$

The resulting state $X(f)$ of the sinusoid is a complex number with a phase angle between 0 and 2π and a magnitude $|X(f)|$. The distribution of frequencies in a signal is called a spectrum. The Fourier Transform is invertible, so the input signal can hence be perfectly reconstructed by integrating each complex sinusoid with the state $X(f)$ in the opposite direction around the unit circle [20]:

$$x(t) = \int_{-\infty}^{\infty} X(f) \cdot e^{j2\pi ft}\,df \tag{2.10}$$

In practice, one would like to calculate the Fourier Transform for discrete signals and frequencies. The discrete Fourier Transform of the kth harmonic for a signal of length N is:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi \frac{nk}{N}} \tag{2.11}$$

The vector of the first N harmonics $\hat{\mathbf{x}}$ can be expressed as a matrix-vector product with the input signal $\mathbf{x}$:

$$\hat{\mathbf{x}} = \mathbf{W}\mathbf{x} \qquad \text{where } \mathbf{W}_{kn} = e^{-j2\pi \frac{nk}{N}} \tag{2.12}$$

The input is multiplied with the matrix $\mathbf{W}$ containing roots of unity (complex points on the unit circle). Optimized implementations utilize vectorized and parallel operations to compute the DFT efficiently. Similarly to the FT any complex series of length N can be perfectly reconstructed using N frequencies of the inverse DFT. The inverse DFT is obtained by multiplying the $\hat{\mathbf{x}}$ vector with the inverse of $\mathbf{W}$ which is its conjugate transposition [20].

The input signal is commonly split into fixed-sized windows, which are then analyzed for frequencies using the Fourier transform. The window size needs to be big enough so the lowest frequencies can be estimated but small enough to detect changes in frequency. The hop length determines the overlap between windows by setting the starting location of the next window as an offset from the previous one.

The convolutional theorem shows that convolution-related operations can be computed as multiplication in the frequency domain. Let $g(t)$ and $h(t)$ be two functions with Fourier Transforms $X(f)$ and $Y(f)$ respectively. Then, the convolution of g and h is given by the product in the frequency domain:

$$\mathcal{F}\{g * h\}(f) = X(f) \cdot Y(f) \tag{2.13}$$

Together with an efficient implementation of the DFT convolution-related problems such as filtering or polynomial multiplication can have reduced time complexity. A naive implementation of the Fourier Transform has $O(n^2)$ time complexity as the whole input signal has to be iterated for every frequency. The Cooley-Tukey Fast Fourier Transform [21] is a divide-and-conquer algorithm that reuses computations from higher frequencies in the lower frequencies. The Fourier Transform is used in conjunction with filtering to remove or extract some ranges of frequencies. It can be used on audio, images, and other types of data, as most modalities can be interpreted as signals. The DFT can be visualized as a Time×Frequency diagram called a spectrogram that displays the magnitudes of the result in each window as pixel intensities [20].

2.3.1.2 Discrete Cosine Transform

The discrete cosine transform is another type of integral transform. It has been used in compression such as JPEG. There are several types of the DCT, each of which differs by boundary conditions. The most common version, DCT-II, is defined as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \tag{2.14}$$

where x_n is the input signal, N is the signal length and k is the frequency. Unlike the DFT, it only works with real-valued inputs and produces a real-valued output.

2.3.1.3 Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCCs) are a common preprocessing technique for sound and acoustic analysis. Parts of the spectrum of a signal are usually very correlated, and it might be inefficient to use this format. MFCCs alleviate this by binning together close frequencies and decorrelating the bins with the DCT.

Calculating MFFCs includes the following steps [22]:

1. Splitting the waveform into smaller segments known as frames, typically with sizes that are powers of 2.
2. Use a windowing function such as the Hanning window to taper the edges of the frames, reducing spectral leakage from low frequencies.
3. Use a DFT implementation such as the Fast Fourier Transform to calculate the power spectrum (the squared norm of the DFT).
4. Bin close frequencies together using triangular filters evenly spaced on the mel scale. See figure 2.5.
5. Take the logarithm of each bin.
6. Use the DCT on the resulting bins as if they were a signal.

This process results in a more feature-dense sequence of numbers compared to the original waveform. Humans do not perceive the frequency nor loudness of sound linearly. The Mel scale was designed to mimic human frequency perception [22].

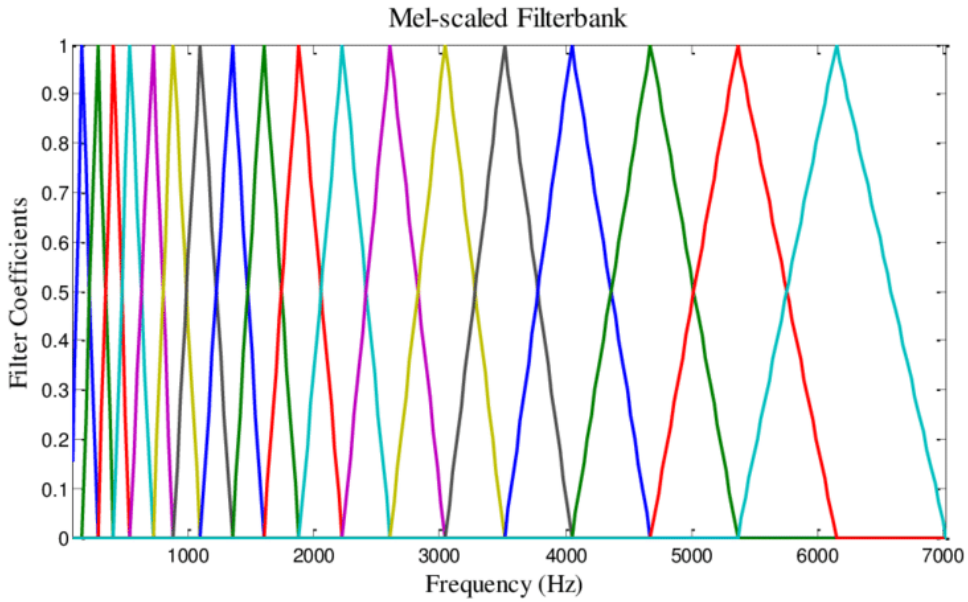

Figure 2.5 20 triangular mel scaled filters, each color is a different filter [23]

2.3.2 Automatic Music Transcription

Automatic Music Transcription (AMT) is a branch of music information retrieval (MIR) that aims to automatically convert audio recordings of musical performances into symbolic representations, such as MIDI files or sheet music. Instrument-specific systems, such as those for automatic piano transcription, focus on transcribing a single instrument.

AMT systems often pre-process the input waveform into the frequency domain by creating a spectrogram, which might be compressed further using MFCCs. A frame in the context of AMT is a part of the input waveform being transformed with the DFT as defined in 2.3.1.1. Frames can overlap depending on the hop length chosen for calculating the spectrogram. The common objectives of AMT systems include determining which notes are active in each frame, when they were turned on and off, and what the velocity of each note is. A basic type of format, the frame-wise binary classification uses a T×F sized output roll, where T is the number of input frames, and F is the number of pitch classes, in this case, 88 for each key on the piano. The transcription system assigns a true or false value (1 or 0) to each pitch and frame if it predicts the note is active. The performance can then be evaluated using standard metrics such as Precision, Recall, and the F measure 2.8. Neural network-based solutions could use binary cross entropy as a loss function. The downside of predicting just the frame-wise activation is that additional post-processing is needed to estimate the onset and offset events. It also does not predict the note's velocity [4].

2.3.3 Regressing Onsets and Offsets Times

Another drawback of predicting onsets and offsets frame-wise is that the frame width and hop length limit the accuracy. A frame with the size of 2048 on a signal with a sampling rate of 16kHz will have a width of 128ms. An onset could happen anywhere in that period but the frame-wise prediction would always be fixed to one point within each frame. The architecture of [4] solves this issue by predicting onsets and offsets with regression inspired by the bounding box prediction of YOLO [18]. The onset/offset prediction targets are inverse normalized distances to the nearest onset/offset event from the center of the frame within a distance of J frames. The exact location of the onset/offset can be calculated from the frame-wise distance predictions during inference. If the middle frame of three consecutive frames is the local maximum and surpasses an onset threshold, it is considered to contain an onset.

Features are extracted from the log mel spectrogram using 4 neural networks for the different tasks of velocity regressions, onset regression, offset regression, and frame classification. Each neural network is a series of convolutional and pooling layers followed by a fully connected layer and bi-directional gated recurrent units [24]. Recurrent units are used to capture the changes in activated notes throughout their duration. Furthermore, the final prediction of onsets is conditioned on the velocity prediction, while the frame classification is conditioned on both the onsets and offsets. The models were subsequently trained on the maestro dataset [25] containing over 200 hours of piano performances with fine alignment between MIDI and audio. Additionally, the onset/offset regression technique was adapted to predict the sustain pedal usage [4].

2.4 Hand Tracking

Hand landmark detection is the process of extracting hand landmarks in 2 or 3 dimensions from an image or depth map.

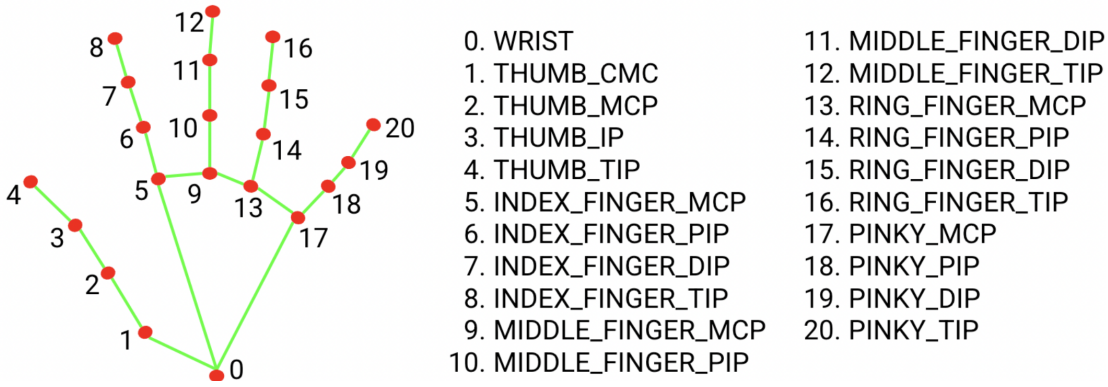

Figure 2.6 Locations of 21 standard hand landmarks [26]

2.4.1 Mediapipe

Mediapipe[26] is a suite of easily accessible libraries for AI-powered applications. The project contains models and supporting code for various tasks and modalities, primarily focusing on CV tasks. The code is maintained in an open-source repository at: `https://github.com/google/mediapipe`. The model architectures and corresponding training data are not always published, but basic metrics and a fairness evaluation are provided in model cards.

Mediapipe provides a solution for hand tracking/hand landmark detection called Mediapipe Hands, described in [26]. The objective of this solution was to allow for real-time hand tracking on mobile devices with standard camera input and minimal sacrifices in accuracy. The pipeline is able to process hand landmarks with a 17.12 ms latency (∼60FPS) on a Google Pixel 6 device. The task is split into two parts, palm detection, and hand pose estimation, each solved by a different neural model.

The palm detection model takes an input frame re-scaled to 256x256 pixels and produces a set of palm bounding box estimates which are subsequently refined with non-maximum suppression 1. The model architecture is based on the Blaze face detection model [27], which in turn contains a modification of SSD (Single shot MultiBox detector) [28]. The Blaze face model is also optimized for latency, and yields results in <6ms on GPUs of selected flagship phones. The bounding box estimates are produced by predicting the difference in location and size between boxes of predefined shapes and aspect ratios on feature maps from selected layers of the feature extractor convolutional neural network. The custom feature extractor makes use of depth-wise separable convolutions to improve the latency [29].

The hand landmarker model takes cropped sections of the input images provided by the palm detector and produces 21 landmarks in 3 dimensions, a left-right handedness, and a hand presence confidence score. If the input frames are from a video or real-time stream, the system will skip the palm detection phase and instead derive the palm bounding box from the landmarks of the previous frame. The models were trained on a combination of in-the-wild, in-house, and synthesized data with different properties and augmentation techniques. The depth dimension was trained purely on synthetic hand images rendered from variations on hand models in different environments created by the GHUM model [30, 31].

Chapter 3

Implementation

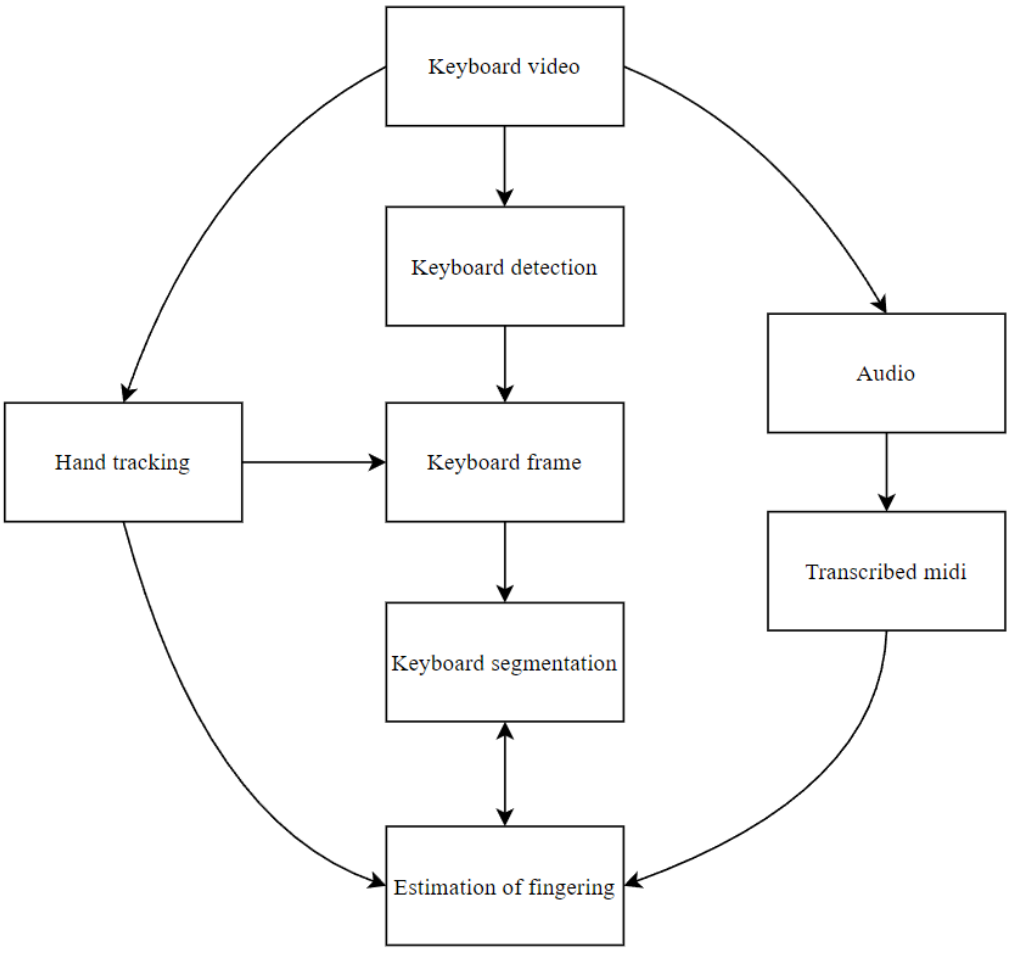

Figure 3.1 A diagram of the pipeline

The data processing pipeline was designed to produce a keyboard fingering estimation from videos with an overhead keyboard view 3.1. Initially, the audio component is extracted from the input video and analyzed for note events using automatic piano transcription [4]. The frames of the video are used for extracting hand landmarks [26] and finding sections that contain a keyboard [18]. If such sections are present, the background keyboard frame without the hands is estimated. The background frame is used to segment and label the individual keys. The fingering is estimated from the hand landmarks and key locations. The key labels can additionally be octave-corrected by minimizing finger distances.

The project is written for Python 3.10 and utilizes several external modules described in the requirements.txt file for computer vision, piano transcription, hand tracking, data manipulation, video processing, etc.. A main *PianoVideo* class was created for interfacing with videos in the overhead format and extracting relevant data points. MIDI transcriptions and the corresponding fingering estimates are internally represented as interval trees for efficient point queries and iteration. Each step in the processing pipeline is saved to the file system, allowing for viewing or re-running only specific steps. The transcription, fingerings, key locations, and keyboard section are stored in a JSON format and can be inspected in a text editor. Background estimates are stored as PNG image files and hand landmarks as a binary array of 32-bit floats.

3.1 Video Collection

The target videos this work is going to extract and process contain footage of someone playing the piano filmed from an overhead perspective as shown in 3.2. The videos can contain different types of pianos, and the camera can be zoomed into different parts. Five YouTube channels were chosen, each containing a significant number of keyboard performances. 100 of the latest videos were downloaded from each channel to create a 500-video development dataset. When possible, videos are acquired at a 30FPS frame rate and 480p resolution; 480p is enough to distinguish individual key bounds at low zoom levels visually.

The type of videos contained in this dataset varies vastly, about half of the videos contain some overhead keyboard frames. Out of those, some are performances, and others are tutorial-style videos with commentary in between playing. Some common variations of this format display cropped musical scores or a falling piano roll above the keyboard. Some videos are vlogs and might not contain any piano music at all, and others contain music filmed from a different perspective or for other instruments.

Figure 3.2 An example frame in the overhead format

3.2 Keyboard Detection and Sectioning

Manually identifying where a keyboard is located is time-consuming and would not be possible for large sets of data. Detecting the keyboard is needed both for validating the format of sections in the video and finding the locations of each key. The detection model should work on a variety of sources, identify sections with a keyboard filmed from an overhead perspective, and provide the bounding box of the keyboard in each section.

To train and test the keyboard detection possibilities, two frames were extracted at random points in each video, creating a set of 990 sample frames, 556 of which contain a keyboard from an overhead angle and 434 do not. The 556 images containing a keyboard were manually annotated with bounding boxes. The frames were resized to fit into a 640x640 pixel boundary and divided into a 33-67% test-train split.

Initially, a multi-scale template matching algorithm was used to match all locations above a certain threshold, followed by clustering. This approach achieved a 98% detection accuracy but was not able to estimate boundaries confidently. A neural network could do both steps simultaneously; the YOLO architecture was chosen for its performance and established documentation. The YOLOv8 small model was fine-tuned on the training data with default parameters. It achieved a 100% detection accuracy and 0.995 mAP score with a 0.5 IOU threshold on the test set; no hyper-parameters were modified. Both the dataset and the model are released with this work.

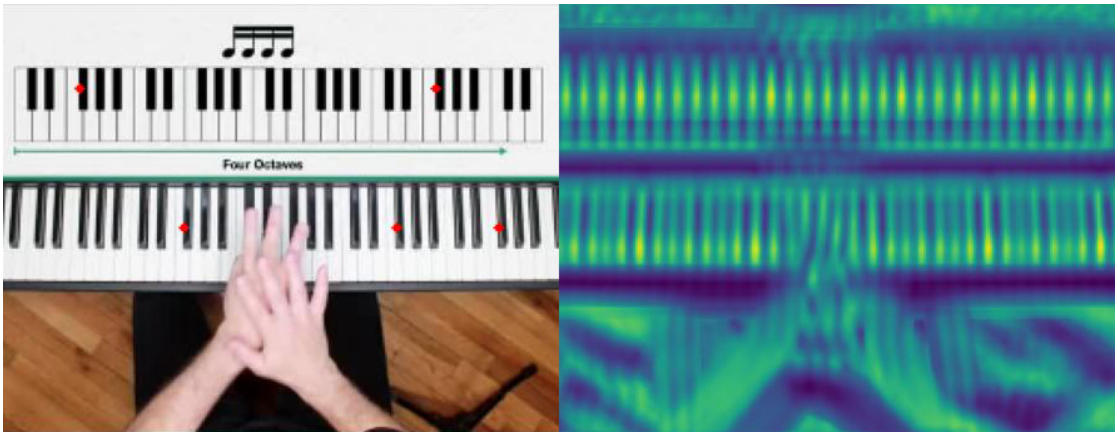

Figure 3.3 Template matching with a small section of the keyboard

Using the keyboard detector, relevant sections of frames in each video were identified. Running the keyboard detection model on every frame of every video was time-consuming and unnecessary. Instead, an approximation algorithm was used. Firstly, sections with a keyboard seldom change every frame, so only every Nth frame is checked for keyboard presence. Additionally, the IOU of keyboard boundaries is compared, if two bounding boxes do not meet a threshold of 0.95 they will belong to different sections. The edges of each section are subsequently refined, needing at most N steps per section. N was chosen to minimize the number of keyboard detection inferences calculated as:

$$\text{inference\_count} = \frac{\text{frame\_count}}{N} + 2N \times \text{section\_count}$$

The average video contains 9012 frames and approximately one section, leading to an inference minimum of $N \approx 67$.

3.3 Background Extraction

The keyboard in detected frames is usually obstructed by hands in some way. These obstructions make the key segmentation process difficult. A background extraction algorithm was devised to extract the background from the sections containing a keyboard. Initially, the median of each color channel and pixel was used as an approximation, this proved to be impractical because the median calculation required storing all frames in memory, and some parts of the keyboard could be obstructed by hands for the majority of the video. Obstructed locations are instead removed from the calculation utilizing the hand landmark data. The background is estimated as the pixel-wise mean over the sections containing a keyboard without the hands. A sum of frames is accumulated with bounds around hand landmarks excluded. By default, the background of the video contains the keyboard scene with the most frames.

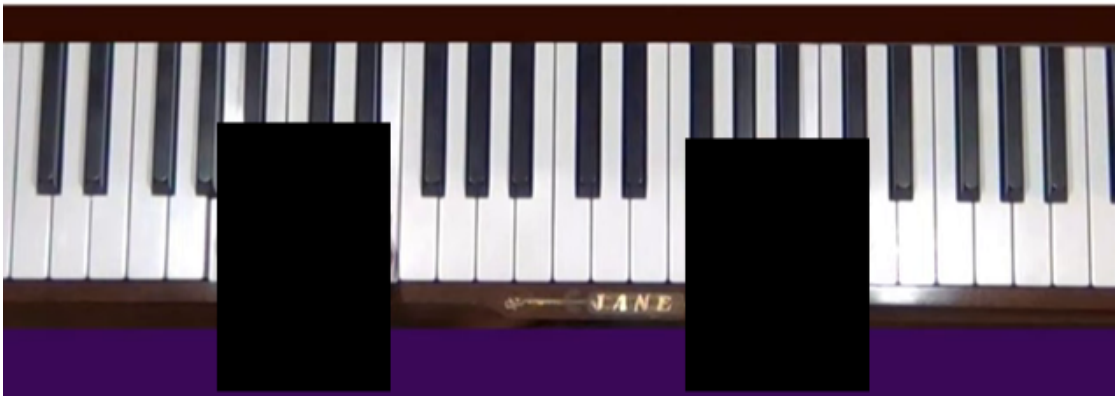

Figure 3.4 Removal of hand locations for background extraction

3.4 Hand Landmarking

Tracking the hands accurately is crucial for estimating interactions with the piano. Most of the videos present are well-lit and should work reasonably well with an off-the-shelf solution. Mediapipe hands [26] provides an efficient hand detection and pose estimation system with 3D landmark output. Some errors arise when using this approach. The handedness prediction is sometimes wrong, causing a few frames of two hands with the same handedness. This was resolved by re-assigning left and right-handedness according to the average horizontal position similar to [3]. Additionally, fast striding movements can cause motion blur, making the algorithm ineffective for a few frames. Gaps smaller than 15 frames are therefore filled in using linear interpolation.

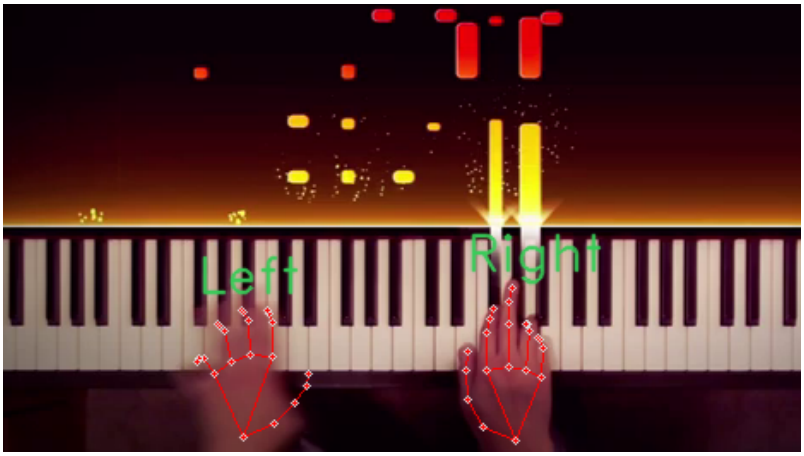

Figure 3.5 Example of hand tracking on a pianist

3.5 Keyboard Segmentation

The key segmentation algorithm receives a background estimate of a video or section and the bounding box of the keyboard. A multi-step process is employed to separate and label each key in the keyboard area.

1. Conversion to the HSL colorspace. The black and white keys vary mostly by lightness.
2. Calculate the mean lightness within the keyboard bounding box.
3. Create an approximate mask of the black keys by thresholding pixels with a lightness smaller than 0.7 times the mean.
4. Estimate the bottom of the black keys as the vertical point where the proportion of black keys is smaller than 0.15.
5. Mark all x coordinates where the mask changes from 0 to 1 as a left and 1 to 0 as a right vertical edge.
6. Match the left and right edges, creating the horizontal bounds for the black keys.
7. Calculate the mean width of the black keys.
8. Mark all gaps between black notes as *large* if they exceed 1.1 of the mean width.
9. Group together black keys bounded by *large* gaps. In most cases, except for the bounds, these groups will have alternating sizes of 2 and 3.
10. From the center-most group, label the notes in each group $C\#$, $D\#$ and $F\#$, $G\#$, $A\#$ alternatingly. A mismatch in group sizes will lead to a partial match of the notes.
11. White key vertical boundaries are derived from the black keys in proportions similar to a standard piano.
12. The C note closest to 0.43 of the background width is considered to be C4. All octaves are calculated in relation to this note. This octave prediction is refined in a post-processing step with hand locations taken into account.

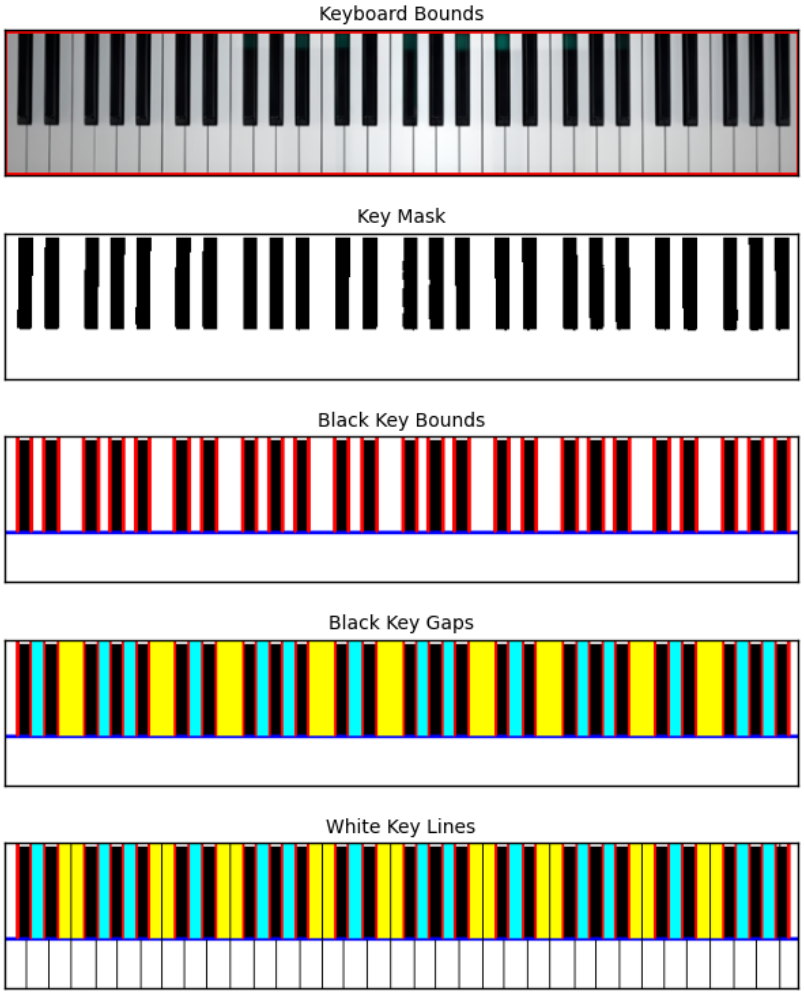

Figure 3.6 Key segmentation steps

Constants and thresholds used in the described process were found by experimentation on the 500 video dataset. Videos of container scenes with zoomed and shifted views of the keyboard. Due to the repeating nature of the keys, it might not be possible to pinpoint the key labels from the background visuals alone. Key label estimates are hence refined using the piano transcription and hand positions. Labels will be shifted up or down by an octave (12 notes) if the sum of minimum key-to-finger distances is minimized.

Figure 3.7 Segmentation comparison on 2 videos each from 5 sources

Chapter 4
Experiments

4.1 Data Properties

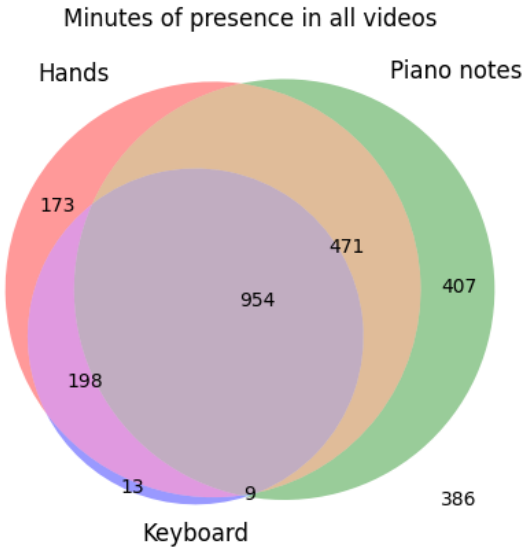

Figure 4.1 Contents of recent videos from select channels

As described in Section 3.1 YouTube channels focused on tutorials and performances filmed from an overhead perspective were chosen for experimentation. The content of these videos was analyzed to assess which fraction of the videos and channels contained useful information for evaluating pianist movements and fingerings. The number of frames containing positive hand landmark predictions, keyboard detections, and note activations was tallied. The 500 videos had a duration of 2611 minutes or 43.5 hours, 954 of these minutes, or over 36% contained all three attributes. This suggests that by merely picking channels focused on a similar format the system can collect tens of hours of video containing structured data relevant to piano performances. On the other hand, 64% of frames are still missing at least one of these aspects, supporting the need for an automatic system to identify and section relevant parts.

Modern pianos have well-established key widths that could be used as references for estimating the sizes and distances of hands in standard units of distance. The white key width of a standard piano is approximately 23.6 mm [32]. Multiplying the standard width with the average white key width estimate from the segmentation algorithm provides an approximate scale reference for each video scene. The accuracy of the reference could be studied by looking at the standard deviations of hand widths for the same performers across multiple videos. The hand size of the same performer should stay relatively stable across scenes.

Hand widths were estimated per video by calculating the mean distance between the most "stable" pair of landmarks. Certain hand landmarks have a lot of movement and variance in distance from other landmarks; it would not be practical to use these as a proxy for hand size. Coefficients of variation were calculated on distances between all landmark pairs on videos in which 50% or more frames contain a keyboard and hands. The distance between landmarks 0 and 14, as shown in Figure 2.6, consistently had the lowest coefficient of variation of approximately 0.059, so they were used to estimate hand size. The per video hand size was calculated as the mean distance between landmarks 0 and 14 multiplied by the scale reference. The distribution of hand widths by channel in figure 4.2 shows that the hand size sometimes varies within a channel. The mean standard deviation of hand sizes within a channel was 6.72 mm while being as low as 2.44 mm for some. In cases such as the *flowkey* channel, videos contain multiple performers that are reflected as clusters in the distribution.

Channel	**Hand size mean**	**Hand size std dev**
Erik C 'Piano Man'	118.75	4.84
Jane	106.08	5.47
Liberty Park Music	121.75	13.13
Paul Barton	121.63	2.44
flowkey – Learn piano	118.66	7.72
Overall	**117.37**	**6.72**

Table 4.1 Mean hand size and standard deviation for each channel

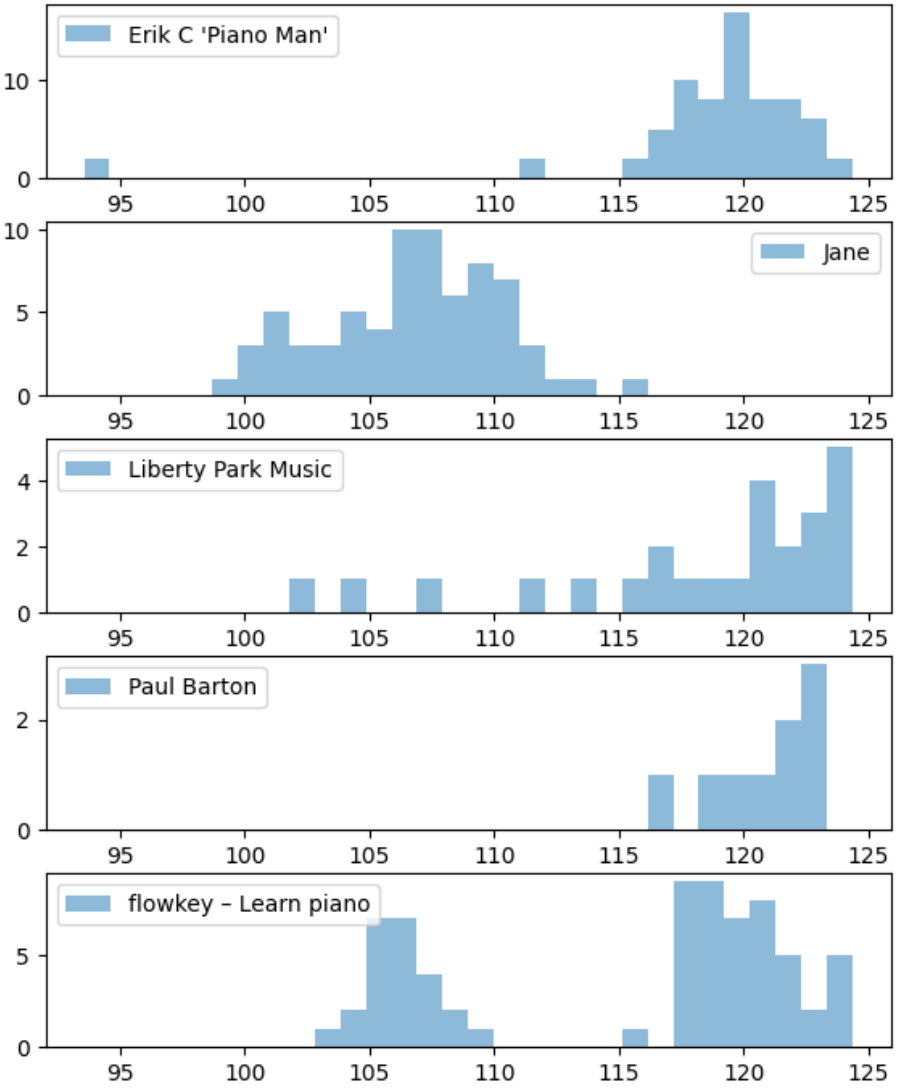

Figure 4.2 Estimated hand size distribution per channel, x-axis is in approximated millimeters

4.2 Keyboard Fingering Estimation

Knowing the positions of the hands and keys allows for estimating the fingering used for playing each note. Experimenting with estimation approaches required an evaluation video with corresponding ground truth MIDI key presses to isolate any errors created by the piano transcription model. Using the built-in microphone and camera on an Android phone a 2,5-minute long video recording was captured on a digital piano together with the MIDI output containing 1227 notes. The ground truth MIDI was aligned to the video by maximizing the F1 score to the piano transcription estimate. Ground truth fingerings were subsequently manually labeled for each of the 1227 notes. Labels were in the format {LR}_{12345} indicating left-right handedness and the finger number. Consistent with other works [1, 3], each note is only assigned a single fingering corresponding to the finger that caused the onset of the note. This does not capture rare cases where the fingering changes during a single note duration.

Datapoints were extracted using the pipeline and a heuristic was tested against the evaluation video. Fingering will be assigned by measuring the finger-key distance between the key currently being played and landmarks on the hands. The distance to the key is calculated from the fingertip landmark F with indices 4, 8, 12, 16, and 20 as shown in Figure 2.6. If the fingertip F is outside the bounding box the distance is calculated as the shortest distance from F to the closest point P on a key bounding box plus the distance from P to the center of the key C. If the fingertip landmark F is inside the bounding box the distance is just the Euclidean distance to the center of the key.

$$D(F,K) = \begin{cases} d(F,C) & \text{if } F \text{ is inside the bounding box} \\ d(F,P) + d(P,C) & \text{if } F \text{ is outside the bounding box} \end{cases} \tag{4.1}$$

where d is the Euclidean distance, F is the fingertip landmark, P is the closest point on a key bounding box, and C is the center of the key.

The finger-key distance is computed between the fingertips on both hands in the frames that contain any note onsets. The finger with the minimal finger-key distance is the fingering label of the pressed note. Comparing the estimates with the ground truth labels of the 1227 notes results in an 82.23% accuracy for the *bounds* metric. The accuracy was also compared for several other distance metrics such as *bounds_0* where the fingertips inside the bounding box are simply assigned the distance of 0. In approximately 10% of the notes two or more fingertips are within the same key bounding box, in these cases, the fingertips with the lowest index are prioritized. Metrics *center* and *bottom center* measure the distance to the center of the key and the center of the bottom edge of the key.

Distance metric	Accuracy (%)
center	39.12
bottom center	51.51
bounds_0	82.07
bounds	82.23

Table 4.2 Fingering estimation accuracy for different distance metrics

4.3 Transcription correction

Figure 4.3 Transcription accuracy vs hand distance threshold

Piano transcription false positives commonly show up in the harmonics of a note [4], i.e. the same notes in different octaves. This type of error could be identified with hand positions as the hands are usually too far away. A distance parameter was added to correct the produced transcription. Notes distanced above a certain threshold from any of the hand landmarks would be deemed erroneous and removed. The threshold distance should be measured in real units, such as the average white key width to enable usage across videos.

Removal of false positives is unlikely to change piano transcription metrics such as those for note onset/offset or velocity evaluation. The transcription correction was therefore evaluated frame-wise. The frame-wise metrics are usually produced for a frame size corresponding to the hop length used in the preprocessing pipeline. In this case, regressing note onsets/offsets were quantized into two different frame sizes 10ms and 33.3ms. 30 FPS (corresponding to a 33.3ms frame size) is the most prevalent frame rate in the channel videos and the evaluation video, in most cases this will be the maximum resolution at which hand landmarks can be obtained accurately. 10ms frames are used in the evaluation of [4], in this case, landmarks from the previous video frame are used for the correction.

Figure 4.3 displays the improvement of the frame-wise F1 score with 33ms frames across different values of the distance threshold. A max distance threshold of approximately 0.34 key widths performs the best on the evaluation video, reaching a top F1 score of 64.2%. This is an increase of 6.6% over the 57.6% without corrections. As seen in Tables 4.3 and 4.4 the improvement comes from a 10%+ increase in precision; recall was slightly lowered as some true positives can be deleted in the process.

	P (%)	R (%)	F1 (%)
No correction	47.95	72.26	57.64
Best correction	58.55	71.11	64.22

Table 4.3 Transcription performance on 33.3ms frames

	P (%)	R (%)	F1 (%)
No correction	47.78	71.90	57.41
Best correction	58.72	71.06	64.30

Table 4.4 Transcription performance on 10ms frames

Chapter 5
Discussion

5.1 Limitations

Although efficient and practical, the mediapipe hands solution is not adapted to piano performance videos. It does not perform well in difficult light conditions and on fast movements. Unpaired image translation was used in [3] to improve the quality before pose estimation. A model could be finetuned on synthesized data [31] in conditions more similar to public videos. The nature of the task allows for using non-real-time systems and possible conditioning from future frames. Occasional errors in handedness should ideally also be solved using the hand-tracking approach and model, not in the output using heuristics.

The transcription accuracy of the test video is lower than the proposed results in [4]; this is likely caused by the quality of the audio input. Further work is needed to extend the ground truth dataset with higher-quality audio sources.

The background extraction algorithm requires the frame to be still, slow drifting motion is in rare cases present and will cause more sections to be created. Dividing sections of keyboard scenes using just IOU is also not perfect, in some cases, the keyboard may change scenes by just moving out of the frame horizontally which would lead to an equal IOU.

The keyboard segmentation task does not function well with strong shadows; a neural network-based segmentation technique would likely resolve this. A tilt in the camera or warping of the lens might also negatively affect the segmentations. The finger estimation format used is simplified although consistent with previous works [1]; only the finger causing the onset is considered. In rare cases, real scores can contain instructions for finger substitution, where the finger changes throughout a note's duration. Even the onset might be caused by more than one finger simultaneously in some situations (glissando).

5.2 Future work

The lack of video recordings with ground truth MIDI files for testing could be remedied by using alignment techniques such as DTW [**MIDI˙dtw**] between video performances and the maestro dataset [25]. MIDI-to-MIDI alignment could also be used to study the movements and fingerings of multiple performers on the same piece. There are usually multiple available video recordings in the correct format for known pieces.

The fingering estimation procedure should be improved to take into account multiple consecutive frames. Some knowledge of past and future notes might be needed to accurately estimate fingerings from fingers occluded by the palm or the other hand.

Recent works have created simulated environments for piano playing with robotics and attempted solutions using reinforcement learning [33]. The reward function is shaped from PIG dataset fingerings [1] and the target MIDI. A human prior, such as the fingering, is likely needed due to the high-dimensional action space [33]. Data from piano videos could be used as a more generally available prior containing both hand motions, fingerings, and MIDI estimates or be used to train a policy directly [34].

Other techniques could utilize this pipeline to produce hand landmarks with motion synthesis [35] just based on MIDI input. Additionally, image or video generation from landmarks could be achieved through motion transfer [36] and conditional image generation.

Chapter 6

Conclusion

In conclusion, this thesis addresses the task of automatically extracting and analyzing piano performance videos filmed from an overhead perspective. Leveraging recent advancements in computer vision and machine learning, particularly in piano transcription and hand tracking. An automatic processing pipeline was created to extract relevant data points, including key and keyboard locations. The usefulness of the data collection approach has been demonstrated by estimating fingerings with an accuracy of up to 82% using a heuristic approach. Furthermore, transcription correction techniques improve frame-wise F1 scores by 6.6%, indicating the potential of refining transcription accuracy with video data. Despite these advancements, several limitations exist, including the need for fine-tuning a hand landmark detection model for piano-specific scenarios, improving testing data, and key segmentation reliability. Source codes for the project are publicly available at `https://github.com/uel/piano-video`. In summary, this thesis represents a first step towards automating the analysis of piano performances using overhead videos, with promising avenues for further research.

Bibliography

1. NAKAMURA, Eita; SAITO, Yasuyuki; YOSHII, Kazuyoshi. Statistical Learning and Estimation of Piano Fingering. *CoRR*. 2019, vol. abs/1904.10237. Available from arXiv: 1904.10237.
2. MACRITCHIE, Jennifer; BAILEY, Nicholas J. Efficient Tracking of Pianists' Finger Movements. *Journal of New Music Research*. 2013, vol. 42, no. 1, pp. 79–95. Available from DOI: 10.1080/09298215.2012.762529.
3. MORYOSSEF, Amit; ELAZAR, Yanai; GOLDBERG, Yoav. *At Your Fingertips: Extracting Piano Fingering Instructions from Videos*. 2023. Available from arXiv: 2303.03745 [cs.CV].
4. KONG, Qiuqiang; LI, Bochen; SONG, Xuchen; WAN, Yuan; WANG, Yuxuan. High-Resolution Piano Transcription With Pedals by Regressing Onset and Offset Times. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 2021, vol. 29, pp. 3707–3717. ISSN 2329-9290. Available from DOI: 10.1109/TASLP.2021.3121991.
5. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. Available from arXiv: 1703.10593 [cs.CV].
6. NEUTELINGS, Izaak. *Neural networks* [online]. tikz, 2022 [visited on 2024-02-06]. Available from: https://tikz.net/neural_networks/.
7. GOODFELLOW, Ian J.; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. http://www.deeplearningbook.org.
8. VERSCHOOF-VAN DER VAART, W.B.; LAMBERS, K. Learning to Look at LiDAR: The Use of R-CNN in the Automated Detection of Archaeological Objects in LiDAR Data from the Netherlands. *Journal of Computer Applications in Archaeology*. 2019, vol. 2, no. 1, pp. 31–40. Available from DOI: 10.5334/jcaa.32.
9. GOWDA, Shreyank N; YUAN, Chun. *ColorNet: Investigating the importance of color spaces for image classification*. 2019. Available from arXiv: 1902.00267 [cs.CV].
10. UNIVERSITY, Stanford. *Template Matching* [https://web.stanford.edu/class/ee368/Handouts/Lectures/2019_Winter/9-TemplateMatching.pdf]. 2019. Accessed: May 10, 2024.
11. JODOIN, Pierre-Marc; MADDALENA, Lucia; PETROSINO, Alfredo; WANG, Yi. Extensive Benchmark and Survey of Modeling Methods for Scene Background Initialization. *IEEE Transactions on Image Processing*. 2017, vol. 26, no. 11, pp. 5244–5256. Available from DOI: 10.1109/TIP.2017.2728181.

12. BOUWMANS, Thierry; JAVED, Sajid; SULTANA, Maryam; JUNG, Soon Ki. *Deep Neural Network Concepts for Background Subtraction: A Systematic Review and Comparative Evaluation*. 2018. Available from arXiv: 1811.05255 [cs.CV].

13. KANG, Dae-Young; DUONG, Pham; PARK, Jung-Chul. Application of Deep Learning in Dentistry and Implantology. *The Korean Academy of Oral and Maxillofacial Implantology*. 2020, vol. 24, pp. 148–181. Available from DOI: 10.32542/implantology.202015.

14. ZHAO, Zhong-Qiu; ZHENG, Peng; XU, Shou-tao; WU, Xindong. *Object Detection with Deep Learning: A Review*. 2019. Available from arXiv: 1807.05511 [cs.CV].

15. MAHDI, Fahad; MOTOKI, Kota; KOBASHI, Syoji. Optimization technique combined with deep learning method for teeth recognition in dental panoramic radiographs. *Scientific Reports*. 2020, vol. 10. Available from DOI: 10.1038/s41598-020-75887-9.

16. CHARLES UNIVERSITY, Faculty of Mathematics; PHYSICS. *Slides for NPFL138* [https://ufal.mff.cuni.cz/~straka/courses/npfl138/2324/slides.pdf]. 2024. Accessed: May 10, 2024.

17. GILG, Johannes; TEEPE, Torben; HERZOG, Fabian; WOLTERS, Philipp; RIGOLL, Gerhard. *Do We Still Need Non-Maximum Suppression? Accurate Confidence Estimates and Implicit Duplication Modeling with IoU-Aware Calibration*. 2023. Available from arXiv: 2309.03110 [cs.CV].

18. REDMON, Joseph; DIVVALA, Santosh Kumar; GIRSHICK, Ross B.; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, vol. abs/1506.02640. Available from arXiv: 1506.02640.

19. REDMON, Joseph; FARHADI, Ali. YOLO9000: Better, Faster, Stronger. *CoRR*. 2016, vol. abs/1612.08242. Available from arXiv: 1612.08242.

20. OSGOOD, Prof. Brad. *EE261 - The Fourier Transform and its Applications*. 2007. Available also from: https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf.

21. COOLEY, James W.; TUKEY, John W. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*. 1965, vol. 19, pp. 297–301. Available also from: https://api.semanticscholar.org/CorpusID:121744946.

22. ABDUL, Zrar Kh.; AL-TALABANI, Abdulbasit K. Mel Frequency Cepstral Coefficient and its Applications: A Review. *IEEE Access*. 2022, vol. 10, pp. 122136–122158. Available from DOI: 10.1109/ACCESS.2022.3223444.

23. ALI, Yusnita mohd; M P, Paulraj; YAACOB, Sazali; YUSUF, Raghad; ABU BAKAR, Shahriman. Analysis of Accent-Sensitive Words in Multi-Resolution Mel-Frequency Cepstral Coefficients for Classification of Accents in Malaysian English. *International Journal of Automotive and Mechanical Engineering*. 2013, vol. 7, pp. 1053–1073. Available from DOI: 10.15282/ijame.7.2012.21.0086.

24. CHO, Kyunghyun; MERRIENBOER, Bart van; BAHDANAU, Dzmitry; BENGIO, Yoshua. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. Available from arXiv: 1409.1259 [cs.CL].

25. HAWTHORNE, Curtis; STASYUK, Andriy; ROBERTS, Adam; SIMON, Ian; HUANG, Cheng-Zhi Anna; DIELEMAN, Sander; ELSEN, Erich; ENGEL, Jesse; ECK, Douglas. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In: *International Conference on Learning Representations*. 2019. Available also from: https://openreview.net/forum?id=r1lYRjC9F7.

26. ZHANG, Fan; BAZAREVSKY, Valentin; VAKUNOV, Andrey; TKACHENKA, Andrei; SUNG, George; CHANG, Chuo-Ling; GRUNDMANN, Matthias. MediaPipe Hands: On-device Real-time Hand Tracking. *CoRR*. 2020, vol. abs/2006.10214. Available from arXiv: 2006.10214.

27. BAZAREVSKY, Valentin; KARTYNNIK, Yury; VAKUNOV, Andrey; RAVEENDRAN, Karthik; GRUNDMANN, Matthias. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. *CoRR*. 2019, vol. abs/1907.05047. Available from arXiv: 1907.05047.

28. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott E.; FU, Cheng-Yang; BERG, Alexander C. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, vol. abs/1512.02325. Available from arXiv: 1512.02325.

29. CHOLLET, François. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR*. 2016, vol. abs/1610.02357. Available from arXiv: 1610.02357.

30. MEDIAPIPE CONTRIBUTORS. *Model Card: Hand Tracking (Lite_Full) with Fairness*. 2021. Available also from: https://storage.googleapis.com/mediapipe-assets/Model%20Card%20Hand%20Tracking%20(Lite_Full)%20with%20Fairness%20Oct%202021.pdf.

31. XU, Hongyi; BAZAVAN, Eduard Gabriel; ZANFIR, Andrei; FREEMAN, William T.; SUKTHANKAR, Rahul; SMINCHISESCU, Cristian. GHUM & GHUML: Generative 3D Human Shape and Articulated Pose Models. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6183–6192. Available from DOI: 10.1109/CVPR42600.2020.00622.

32. BOYLE, Rhonda. The Benefits of Reduced-Size Piano Keyboards for Smaller Handed Pianists: An Exploration of Biomechanical and Physiological Factors. In: *Proceedings of the 11th Australasian Piano Pedagogy Conference, Opening Doors: The Complete Musician in a Digital Age*. University of Southern Queensland, Toowoomba, 2013.

33. ZAKKA, Kevin; WU, Philipp; SMITH, Laura; GILEADI, Nimrod; HOWELL, Taylor; PENG, Xue Bin; SINGH, Sumeet; TASSA, Yuval; FLORENCE, Pete; ZENG, Andy; ABBEEL, Pieter. *RoboPianist: Dexterous Piano Playing with Deep Reinforcement Learning*. 2023. Available from arXiv: 2304.04150 [cs.RO].

34. HO, Jonathan; ERMON, Stefano. *Generative Adversarial Imitation Learning*. 2016. Available from arXiv: 1606.03476 [cs.LG].

35. ZHU, Wentao; MA, Xiaoxuan; RO, Dongwoo; CI, Hai; ZHANG, Jinlu; SHI, Jiaxin; GAO, Feng; TIAN, Qi; WANG, Yizhou. *Human Motion Generation: A Survey*. 2023. Available from arXiv: 2307.10894 [cs.CV].

36. CHAN, Caroline; GINOSAR, Shiry; ZHOU, Tinghui; EFROS, Alexei A. *Everybody Dance Now*. 2019. Available from arXiv: 1808.07371 [cs.GR].

Attachment contents

- `readme.txt` — brief description of media contents
- `text` — thesis text
  - `thesis.pdf` — thesis text in PDF format
- `src`
  - `thesis` — source of the thesis in LaTeX format
  - `impl` — source code of implementation
    - `evaluation` — code for experiments
    - `figures` — code for figures in text
    - `src` — pipeline source
    - `notebooks` — keyboard detection training
    - `models` — keyboard detection model