12-19-2018

# An Application of Deep-Learning to Understand Human Perception of Art

Sanjana Kapisthalam
sxk9196@rit.edu

# An Application of Deep-Learning to Understand Human Perception of Art

by

Sanjana Kapisthalam

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Chester F. Carlson Center for Imaging Science
College of Science
Rochester Institute of Technology

Dec 19, 2018

Signature of the Author ____________________

Accepted by ____________________
Coordinator, M.S. Degree Program Date

CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE
COLLEGE OF SCIENCE
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

---

M.S. DEGREE THESIS

---

The M.S. Degree Thesis of Sanjana Kapisthalam
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
M.S. degree in Imaging Science

______________________________
Dr. Elena Fedorovskaya, Thesis Advisor

______________________________
Dr. Christopher Kanan

______________________________
Dr. Andrew M. Herbert

______________________________
Date

An Application of Deep-Learning to Understand Human Perception of Art

by

Sanjana Kapisthalam

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

## Abstract

Eye movement patterns are known to differ when looking at stimuli given a different task, but less is known about how these patterns change as a function of expertise. When a particular visual pattern is viewed, a particular sequence of eye movements are executed and this sequence is defined as scanpath. In this work we made an attempt to answer the question, “Do art novices and experts look at paintings differently?” If they do, we should be able to discriminate between the two groups using machine learning applied to their scanpaths. This can be done using algorithms for Multi-Fixation Pattern Analyses (MFPA). MFPA is a family of machine learning algorithms for making inferences about people from their gaze patterns. MFPA and related approaches have been widely used to study viewing behavior while performing visual tasks, but earlier approaches only used gaze position (x, y) information with duration and temporal order and not the actual visual features in the image. In this work, we extend MFPA algorithms to use visual features in trying to answer a question that has been overlooked by most early studies, i.e. if there is a difference found between experts and novices, how different are their viewing patterns and do these differences exist for both low- and high-level image features. To address this, we combined MFPA with a deep Convolutional Neural Network (CNN). Instead of converting a trial’s 2-D fixation positions into Fisher Vectors, we extracted image features surrounding the fixations using a deep CNN and turn them into Fisher Vectors for a trial. The Fisher Vector is an image representation obtained by pooling local image features. It is frequently used as a global image descriptor in visual classification. We call this approach MFPA-CNN. While CNNs have been previously used to recognize and classify objects from paintings, this work goes the extra step to study human perception of paintings. Ours is the first attempt to use MFPA and CNNs to study the viewing patterns of the subjects in the field of art.

If our approach is successful in differentiating novices from experts with and without instructions when both low- and high-level CNN image features were used, we could then demonstrate that novices and experts view art differently. The outcome of this study could be then used to further investigate what image features the subjects are concentrating on. We expect this work to influence further research in image perception and experimental aesthetics.

## Acknowledgements

I would like to thank my advisor, Dr. Elena Fedorovskaya, for all the help and advice during my studies at Rochester Institute of Technology. Many people have contributed in various ways to make this M.S. study an exciting and a memorable journey. Over the past two years, it has been my great honor to work with all the members of the Discover Lab. I am indebted to the members of my committee, specifically Dr. Kanan and Dr. Herbert, for their input, patience, and support throughout my research. I owe sincere gratitude to CIS, especially, Dr. Messinger, Dr. Kerekes and Dr. Bachmann who advised and supported me in all possible ways to complete my study successfully. In addition, I would like to thank the staff of the Carlson Center for Imaging Science for their support and assistance. I am grateful to my family: my grandparents, parents, and my brothers for always encouraging me and pushing me towards achieving my dreams. Finally, I would like to thank all my friends, near and far, for their love, support, and care.

*to my grandparents*

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Recording eye movements and analyzing them provides valuable information about how people view and comprehend the world. Since the time of research done by pioneers like Buswell and Yarbus, the advances in eye-tracking technology has come very far. Modern remote and wearable eye-tracking devices collect eye movement data non-intrusively. A vast amount of literature is available on studying eye movements with eye-tracking technology, and a wide range of disciplines use these devices including psychology, psycholinguistics, computer science, marketing, etc. The application of eye-tracking has provided more access to vision scientists to delve deeper into studying human vision and visual attention.

Buswell and Yarbus were amongst the first to investigate the relationship between eye movements and high-level cognitive factors [2, 3]. The results from Yarbus' early experiments suggest that human viewing behavior is affected if a task is given while the observer is viewing the stimulus. Since then, many researchers conducted experiments involving tasks with different instructions and provided results supporting Yarbus' early work [4, 5, 6, 7].

While there is a considerable amount of work being done by researchers to study and understand eye movements during various visual tasks [8, 9], there has been an increasing interest in using gaze data in experimental aesthetics to understand the aesthetic experience [10, 11, 12, 13]. It has been shown that differences in gaze patterns while viewing art can be studied by examining fixations, saccades, and scanpaths [14].

A number of researchers attempted to study the perception of art with recorded eye movements [15, 16, 17, 18, 19, 14, 20, 21, 22], to get a comparison between art experts and novices. These studies show that there is indeed a difference between eye movements of experts and novices when viewing images of art and some of them even show that there were striking differences when instructions were given as part of their experiments. Although all these studies were able to demonstrate the differences between art experts and novices viewing art the approaches used by them were mostly descriptive. In [11] it was pointed out, that experts were more likely to have saccades of a larger amplitude compared to novices. The authors claim that the experts were more attentive to the relationships between objects in a painting while on the other hand, novices were attracted to recognizable objects in a painting. These studies are limited in that they cannot predictably link the viewers' expertise or instruction with the gaze data and provide information on the contribution of low- and high-level features of an image to the viewing behavior of the expert and novice observers expressed in their eye movement patterns.

Although, the above stated work shows that there are differences between experts and novices viewing art, the present work aims to answer the following questions: Do experts and novices view paintings differently? If differences exist between their eye movements, what factors contribute to these differences? Do these patterns of eye movements differ based on experience, knowledge, instructions, and image features?.

While we are not the first ones to address these questions, the novelty behind our work stands in our approach to answer them.

For this research, we consider experts as those who studied art history or are trained to understand art in the form of paintings and novices as those who do not have any background in art. In our approach we use machine learning techniques, and the ideas behind our method of analysis are inspired by [23]. We attempt to address our research questions using algorithms, specifically, the methods that employ convolutional neural networks. To the best of our knowledge, we are the first ones to use these methods in the field of experimental aesthetics and art perception, especially in comparing viewing behavior of experts and novices.

# Chapter 2

# Background

Every time we open our eyes, light from the surroundings passes through the cornea, pupil, lens and then finally reaches the retina. This light carries much information. Post retinal processing includes passing all this information from ganglion cells traveling through optic nerves, passing by the Lateral Geniculate Nucleus (LGN), and eventually reaching our visual cortex. Our brain is powerful enough to process this information seen through the eyes in a fraction of a second. This entire process is called “Visual Processing.” It has been said that we are under the impression that we can process an entire visual field in a single glance but the reality is, we cannot process this information outside of the foveal vision if we are unable to move our eyes [24].

## 2.1 Eye Movements and Eye-Tracking Technology

Eye movement is defined as the intentional or unintentional motion of the eye, helping in scanning, fixating, and tracking a visual stimulus. Eye movement is essential to processing details of any information in the visual field because of acuity limitations in our retina. High visual acuity is constrained to the fovea, the small circular region in the central retina, which is a rod free region densely packed with cone photoreceptors. Eye movements are important because they are capable of directing the fovea to new locations of interest or recompense for any disturbances that displace the fovea from the location it is already attending.

Several decades ago, a Russian physiologist named Alfred Yarbus demonstrated that eye movements disclose a great deal about the strategies used while observing a scene. As part of his experimental set-up, Yarbus used contact lenses with mirrors on them to record eye movement patterns of subjects viewing different objects and scenes. In one of his early experiments, he made his observers view a portrait at different times. His analysis demonstrated that when the same image is shown repeatedly, the observers' eye movements are similar but not identical [3]. The results from Yarbus' experiments and many other researchers over the years showed that vision is an active process with eye movements typically shifting from one location to another while viewing a scene to observe especially interesting and meaningful features.

The importance of eye movements in pattern recognition was demonstrated by [8]. They conducted an experiment to record eye movements of their subjects viewing arrays of randomly generated black and white pixels. While viewing the stimuli, the subjects were asked to indicate whether there were identical arrays or differed by one pixel under two conditions. In the first condition, the subjects were instructed to use normal eye movements, and in the second one, subjects were instructed to fixate on a point between the arrays.

When the subjects moved their eyes freely, they quickly found the different pixel in the arrays. However, under the second condition when the subjects were required to look at a fixation point, search for the different pixel was less efficient and sometimes not found despite giving the subjects more time to see the stimulus. The authors' findings showed that eye movements indeed play an essential role in the analysis and recognition of patterns.

More recently, several studies conducted experiments and showed that an observer's task and mental state could be decoded from their eye-movements [6, 7, 23]. All these experiments point in one direction as to how essential eye movements are and how much information one can gather just by analyzing eye movements.

### 2.1.1 Types of Eye Movements

There are four basic types of eye movements - saccades, smooth pursuit movements, vergence movements, and vestibulo-ocular movements. Saccades are rapid, ballistic movements of the eyes which help the fovea to move from one location to another. The slow tracking movements of the eyes that keep a moving stimulus on the fovea are called smooth pursuit movements. Vergence movements help in the alignment of the fovea of each eye with targets located at different distances from the viewer. Vestibulo-ocular movements stabilize the eyes relative to the external world and thus compensate for any head movements.

Eye movements allow us to scan the visual field, to focus our attention, and to pause on areas of the scene that carries most significant information. Tracking these eye movements could potentially tell us what aspects of the scene are of interest to the viewer. For this, we have eye-tracking devices that enable us to track eye movements and provide us with a lot more information in great detail.

### 2.1.2 Recording Eye Movements

Eye-trackers have been used to study eye movements and understand the scientific basis of visual experience. Eye-trackers are devices that enable us to record eye movements and study human viewing behavior concerning the position of the eyes and the movements they make. In basic terminology, measurement of eye activity is called eye tracking. The data from an eye-tracker is collected using either a remote or head-mounted eye-tracker connected to a computer. There is an intrusive and a non-intrusive type of eye-trackers. In the present study, we use a non-intrusive remote eye-tracker, developed by the company SensoMotoric Instruments (SMI). The non-intrusive remote eye-tracker works by illuminating the eyes using an infrared light source and tracks the reflection of the light source in the eyes and pupils with the built-in camera.

The data obtained from an eye-tracker holds the record of fixations and saccades on a scene. Fixations are the relatively stable positions of the eye on a visual stimulus from where visual information is gathered. The sequence of the eye movements organized in time is called saccades, and they help to move from one fixation point to another. Additionally, the eye-trackers provide information on the frequency of blinks and the change in diameter of the pupil which we do not take into consideration as part of our research.

#### Scanpath

Scanpaths hold the record of fixation locations traced by an eye-tracker. Noton and Stark were the first ones to define the concept of a "scanpath" [25, 26]. They proposed the "scanpath theory," according to which when a particular visual pattern is viewed, a particular sequence of eye movements is executed and furthermore, that this sequence is crucial in assessing the visual memory for this pattern. Over the years, this theory has been disputed strongly by researchers who accumulated considerable evidence from experiments with natural images that the human gaze selects informative details, i.e. that eye fixations tend to get drawn towards or near important details of images. However, the formulation of a scanpath as a sequence of fixations along a viewing trajectory has been useful to analyze the patterns of eye movements.

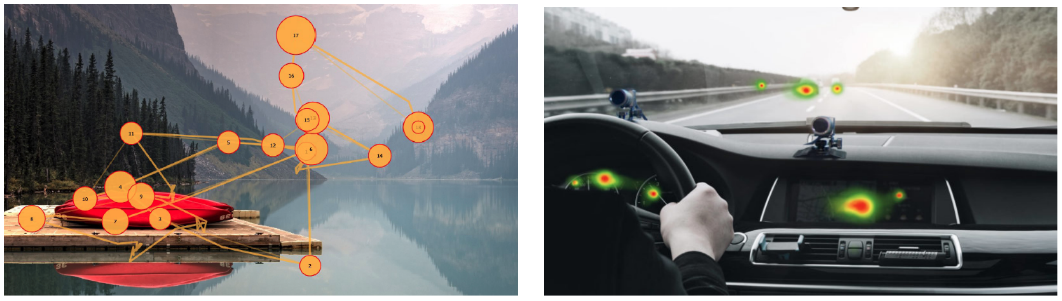

Figure 2.1: Pictorial representation of visualizations obtained from an eye-tracker. On the left is an image which represents a gaze plot where the circles represent the location, its size represents the duration, and the number represents the order in which they traced those locations. On the right is a heatmap which represents which areas of the image attract more attention compared to the others. Image Courtesy - [1].

Throughout our work, we will be considering scanpath as eye movement data recorded by an eye-tracking device where this recorded information is about the scanning path of a visual scene organized in time. Therefore, the scanpath consists of where observers looked at, i.e. their fixation locations, and how long their eyes remained in that location, i.e. duration of fixations. Eye tracking software programs can visualize eye movement data as gaze plots and heat maps (see Figure 2.1). Gaze plot can be considered as a visualization of a scanpath.

## 2.2 Factors Influencing Viewing Behavior

Various factors influence viewing behavior. As part of our work, we are concerned with the influence of task at hand, expertise, and instruction on viewing behavior. In the following sections, we discuss each of these factors briefly.

### 2.2.1 Influence of Tasks on Viewing Behavior

Extensive research has been done in the past to understand human visual perception and comprehension along with the relationship between perceptual and cognitive factors and gaze patterns. The first attempts to study this link were made in [2] and [3]. Yarbus, in his book [3] explains image perception, the basic mechanisms behind human eye movements, and the theory on the role these eye movements play in the processing of visual information. To understand the nature of the gaze patterns while viewing visual stimuli observed in the studies done by [2] and [3], [4] replicated Yarbus' experiment using a head-free eye tracker and described in detail how viewing patterns differ in people and vary if specific instructions are given a priori. Overall, this confirms the seminal work done by [3] and shows that eye movement patterns are task and instruction dependent.

Similar to [4, 3], an experiment was conducted by [5] where the subjects were shown an image of "Alfred Yarbus." There were two conditions imposed, first was a free viewing condition and the second, a condition with a series of changing instructions. Under the free viewing condition, all the subjects fixated at the center, mostly around Yarbus's face. When the participants were given instructions asking them to remember various areas of the image, their fixations were more evenly distributed, focusing on the regions they were asked to remember. This experiment shows that people view the same image differently depending on the instruction given before the experiment. This outcome further confirmed the findings of [4].

The findings from [4, 5] are important for our work as we would like to see if prior instructions affect the viewing behavior of the subjects as part of our experiments.

Presently, two hypothetical mechanisms are used to explain the relationship between gaze patterns, attention, and visual perception: (I)where the image presents specific information relevant to the task at hand, human vision is influenced by top-down factors, i.e., human vision tends to get attracted towards areas of the image with relevant and meaningful information; (II)where the image does not carry essential or useful information available, human vision is influenced by bottom-up factors, i.e., it tends to get attracted towards low-level image features such as color, edges, etc., [27, 28, 29]. Although the hypotheses point toward different mechanisms, they are both considered to be true depending on various factors such as the task at hand, subjects' experience, their a priori knowledge about the image, the type of images being viewed, etc. For example, if a viewer sees a visual stimulus with a goal of searching something in the stimulus or if they are given a task to remember what they saw, this points toward the top-down mechanism. On the other hand, if a viewer merely sees a stimulus with no inner intention or a given task, then it points toward the bottom-up mechanism of visual attention.

### 2.2.2 Influence of Training on Viewing Behavior

An increased interest in studying eye movements has prompted researchers to study the influence of training on viewing behavior. This interest has resulted in many studies leading to the comparison of eye movements of experts and novices in different fields of expertise. For example, for example, [30] showed that expert radiologists had a "gestalt-like" perception while scanning mammography images in detecting breast cancer. In a game like chess, [31] showed that experts fixated beside their next chess moves and all around the center of the board while the novices fixated more often on the pieces they felt the need to move or to protect. As part of this work, the authors mention that this particular eye movement pattern of chess experts was accompanied by bi-lateral activation of the brain while in novices there was only left hemisphere activation. The authors reported that the activation of right hemisphere could be linked to holistic processing of the stimuli. An attempt was made to compare global and local temporal eye movement patterns of experts and novices viewing dermatological images [32]. Their findings show that novices repeatedly fixate on the same location while on the other hand experts do not tend to do this and their fixations are also widely spread in time. The aforementioned scientific studies show that experts indeed have a different viewing behavior when compared to novices and this is because they are trained to do so.

## 2.3 The Role of Art Training in Aesthetic Appreciation

Viewing behavior is said to be affected by salient visual features and cognitive factors such as task at hand [2, 3, 4]. This effect of saliency applies to the images of paintings where the observer's eye is quickly drawn towards recognizable figures, especially faces. People who attend art schools involving classes for art history, painting restoration, drawing, recreating art, etc. are trained to pay close attention to the aspects of the paintings beyond recognizable salient objects. These aspects could be the historical context, styles of the paintings, etc. Hence, artists and experts on art are expected to view paintings differently from novices. This expectation potentially raises many questions to the scientists in the field of vision and perception who are interested in understanding the relationship between art training and its influence on viewing behavior. Some immediate questions one can ask are, how is perception influenced with art training? Does this change the scanning of visual compositions and the information being processed during perceptual analysis? Moreover, does this behavior contribute to aesthetic judgments?

Although early research done by [2, 33] suggested that there are differences between art experts and novices, the authors could not quantify the nature of these differences. Yarbus was among the pioneering researchers who made an attempt to address this and conducted experiments where he recorded the eye movements of his subjects viewing paintings but his studies were mainly centered around questions: One, how is scanning behavior affected while viewing faces? Two, given instruction how does viewing behavior change while inspecting more complex scenes of paintings? Because of his work mainly surrounding these two questions, Yarbus did not intend to quantify the differences in viewing behavior of people in terms of expertise, specifically in the field of art. Nevertheless, Yarbus was able to show how instructions significantly affected the viewing behavior of the subjects.

Since Buswell, Brandt, and Yarbus, the work done by [34] is believed to have come closest in quantifying the role of eye movement patterns to evaluate visual compositions of artworks. In this work, the author assumes that observers fixate on areas of images either to gain knowledge or for pleasure. When the observer fixates to gain knowledge, the eye movements are reported to be slower and deliberate than when they view for pleasure. The author was able to compare visual compositions by calculating the spatial density of fixations spread across the painting and argued that training could make viewers interested in evaluating artworks' visual compositions.

## 2.4 Methods for Analyzing Eye Movements

There are many approaches to analyze eye movement data, ranging from more straightforward techniques evaluating fixation densities and length of saccades to more complex ones involving machine learning. Some earlier methods include variance analysis and recurrence quantification analysis. These previous approaches solely concentrated on fixation data and ignored visual features extracted from images. The advantage of incorporating image features along with fixation data would allow us to understand which areas of the image are being fixated and how these fixated locations related to specific image features confined to those locations.

More recently, with increasing interest in eye movement analysis, many powerful techniques have been developed that explore the statistical properties of eye-movement patterns and relate these with the characteristics of the visual stimulus. Some of these recent techniques include algorithms such as multi-fixation pattern analysis (MFPA), Deep-Fix, SalGan, Encoder-Decoder neural networks [23, 35, 36].

MFPA is a type of algorithm that takes in fixation locations and converts them to essential features and classifies them. Previously, MFPA was used to infer a task which was given to an observer while viewing scenes [6, 7, 23]. The algorithms under the family of MFPA take an observer's scanpath, a sequence of fixations as input, and uses it to infer traits such as the task the observer is given. If the algorithms can make this inference above chance when trained on an observer's scanpath for specific tasks, then it suggests that the observer might have a routine for scanpaths while accomplishing one or more tasks. In the next chapters, we explain this algorithm in more detail.

Deep-Fix, SalGan, and Encoder-Decoder networks are models involving convolutional neural networks to predict human eye fixations. These models are designed for predicting accurate saliency maps and can help to understand human visual attention mechanisms. A saliency map is obtained from an image showing unique quality of each pixel, and the attention mechanism in humans can be defined as the "active direction of the mind to an object." Saliency models were first introduced by [28]. These maps are used to understand human vision by simplifying the representation of where people looked at in an image into something that is more meaningful and easier to analyze. Unlike classic saliency model described in [28] which predicts saliency maps using hand-crafted features, these models hierarchically learn features and predict saliency maps. For this research, we do not implement any of these algorithms but, it could be of potential interest to consider these for future work.

## 2.5 Studies Differentiating Art Experts and Novices

Individual differences in viewing patterns have been researched and studied by carefully examining the fixations of individuals who differ from one another in terms of background and experience. Buswell was the first to study this. In his book [2], he speaks about how people look at pictures. His findings show that most of people's perception comes from peripheral vision. Regardless of the intentions of the artists', Buswell showed that people follow a pattern of perception while viewing paintings. Since then, many researchers worked along the same lines trying to differentiate art experts and novices while viewing images of paintings [15, 16, 17, 18, 19, 14, 20, 21, 22].

It was found that gaze patterns by themselves could differentiate novices and experts by fixation densities on aspects of images of paintings with less importance, where this "importance" was defined by one expert artist and not the experimenters [37]. The authors of this work define a total number of fixations on an image as fixation densities and performed discriminant analysis to explain individual differences. When they showed paintings like

American Gothic, Mona Lisa, and The Last Supper, their subjects did not carefully examine all aspects of these paintings. However, for other paintings, they examine the paintings in a more detailed way. The authors relate this viewing behavior to the training of the expert participants, i.e. the authors claim that the mentioned paintings were more familiar to their expert subjects and hence the subjects did not find a need to examine them. Whereas, for the other unfamiliar paintings they examined more carefully to understand the subtleties of the paintings. It is unclear from their work as to what they mean by unfamiliar paintings since they do not mention what they were.

In [15] fixation data of art experts and novices were examined to determine if altering the balance of the original composition affected the viewing behavior as revealed by the distributions of short and long fixation durations and which areas of the paintings received greatest visual attention. They analyzed the areas in images of paintings, processed by different and specific exploration defined by short- and long-gaze durations (i.e., fixations with less than 300 ms and fixations with greater than 400 ms gaze durations). They showed that novices fixate more on central and foreground areas. While, on the other hand, experts tend to spend more time viewing background areas. Their findings stood by the idea that novices focus more on the individual objects in an image and experts more on relationships between the objects in general. This tells us that training affects viewing behavior in a way that experts scrutinize the paintings much more than novices.

An attempt was made by [16] to differentiate between art experts and novices viewing only realistic and abstract images of paintings. They did variance analysis that showed global/local eye movement indices, were lower for local scanning by art experts who contributed to more global viewing, particularly in abstract images. Novices, on the other hand, carried their local scanpath patterns from realistic images on to abstract images. These indices were calculated as a ratio of long and short amplitude saccades. The blink rate of expert participants viewing abstract images was significantly lower when compared to realistic images, showing an increase in visual effort while novices showed no difference in blink rates.

The results from [17, 19] showed that the frequency and duration of fixations were identical in both the groups of participants. However, the authors claim that the viewing patterns were different in the two groups showing that novices spent more time on the areas of paintings which have recognizable objects and human features when compared to experts. The responses of art experts and novices were compared by [18] when the subjects were shown slide-projected and computer-generated images of nine paintings, and it was concluded that there was a significant difference in the subjects' responses based on expertise.

Findings from [14] show that the viewing patterns of the experts changed with a change in the level of abstraction of paintings. The authors from [20] evaluated the differences using correlation coefficients (CCs) between fixations and saliency maps and show that the CCs of experts were lower in free-viewing conditions, concluding that the viewing patterns of experts depended on their expertise only under a free-viewing condition. Furthermore, it has been shown that art experts are less driven towards low-level features [21].

Recently, [22] studied the difference between art experts and novices for various types of paintings using Recurrence Quantification Analysis (RQA). RQA is a method that uses fixation locations and durations to determine if there are any repetitions in the viewing patterns of the subjects. To be more specific, this method is capable in evaluating if a subject is fixating on the same areas of an image, and how often the subjects go back and forth between these locations. The results from [22] suggest that novices tend to have more consecutive fixations, i.e. some repetitive gaze patterns while viewing paintings when compared to experts. This study also confirms that expertise, instructions, and the type of paintings influence the viewing patterns of the participants.

## 2.6 Machine Learning to Analyze Eye Movements

While all the studies mentioned above and their findings act as a background for our research, we build our work on the conclusions of [22] and [21] by modifying multi-fixation pattern analysis (MFPA) [23] to include CNN features. MFPA is a family of machine learning algorithms for making inferences about people from their gaze patterns. MFPA and related approaches have been widely used to make inferences from subjects, but previous approaches only use gaze position (x, y) information with duration and temporal order and not the visual features in the image [38, 7, 39, 6].

To address this, we combined MFPA with a deep CNN. Instead of converting a trial's 2-D fixation positions into Fisher Vectors, we extracted image features surrounding the fixations using a deep CNN and turned them into Fisher Vectors for a trial. We call this approach MFPA-CNN. If MFPA algorithms can make an inference above chance, this suggests that there are differences between scanpaths for the variable of interest [23].

In [23], MFPA was used to determine whether people had different scanpath routines for making judgements about faces. Here, we use MFPA to determine whether art experts and novices look at art in the form of digital paintings differently under two different conditions: instructed and non-instructed. While CNN's have been previously used to recognize and classify objects from paintings [40, 41], this work goes the extra step to study the human perception of digital paintings as opposed to natural digital images.

If MFPA can infer whether the viewer is an art expert or novice, then we can conclude that art experts have different viewing patterns than novices. Likewise, if we can differentiate instructed from non-instructed participants, then we can conclude that instructions indeed influence the viewing behavior of our subjects. By comparing the models trained on low- and high-level image features, we can gain additional insights about differences seen with different features. If there's difference seen with low-level features, we could argue that our subjects' vision is more driven towards low-level image features like edges, color, intensity, etc. In contrast if difference exists even with high-level features, we can argue that the subject's vision is driven towards high-level features of an image like faces, objects, etc. Note that we call our model trained on low-level features as MFPA-CNN(L) and with high-level features as MFPA-CNN(H).

# Chapter 3

# Study Description

## 3.1 Experiment Design

[22] [22] did the data collection. They followed a 2x2x5 between-subjects experimental design with two expertise levels (experts and novices), two instruction levels (free viewing and a follow-up questionnaire instruction), and five painting categories. The art expert group consisted of 24 students and faculty members, who were studying and/or teaching art and art-related disciplines. The novice group also consisted of 24 students and faculty from various majors with no background in visual art studies and those who were not specifically interested in paintings or any art form. The participants for both groups were recruited from Rochester Institute of Technology and included both female and male genders. The participants were screened for normal color vision and normal or corrected to normal visual acuity. The study has received RIT IRB approval.

## 3.2 Stimuli and Materials

The stimuli consisted of 60 high-resolution digital images of fine art paintings representing five different categories: Abstract, Landscape, Portrait, Cityscape, and Stilllife. Each category had 12 paintings. The SMI Red-250 eye tracker was used with the dual display setup where the images were shown on the full screen of one computer. The second computer was used to run the SMI software as shown in Figure 3.1.

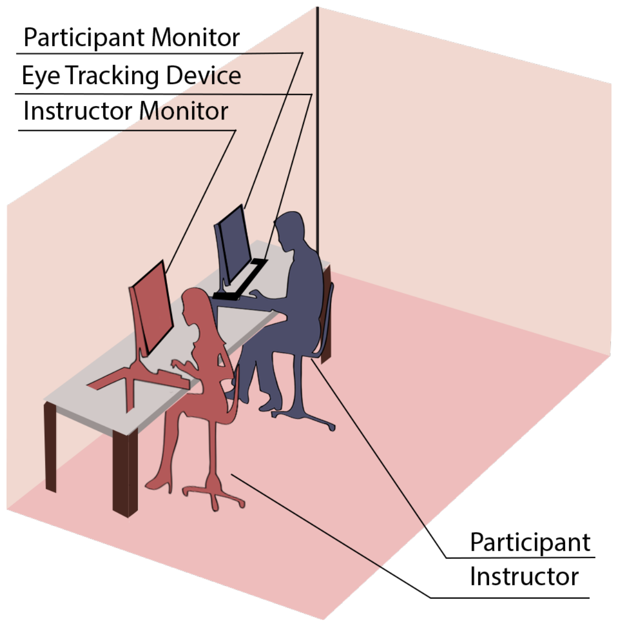

Figure 3.1: Pictorial representation of the experimental set-up.

## 3.3 Procedure

The participants sat at the distance of 60cm from a 50.8cm (20") display where the images were presented on a 1680x1050 resolution screen. Before beginning the experiment, the participants were informed that their eye movements would be recorded. The participants from the instructed group were told that they would be given a questionnaire at the end of the experiment on their understanding and impressions of paintings while those from the non-instructed group were asked to view the images freely. The participants were told to view images as long as they wanted and to say “next" when they wanted to proceed to the next painting. The images were displayed in a randomized order for every participant.

# Chapter 4

# Methods

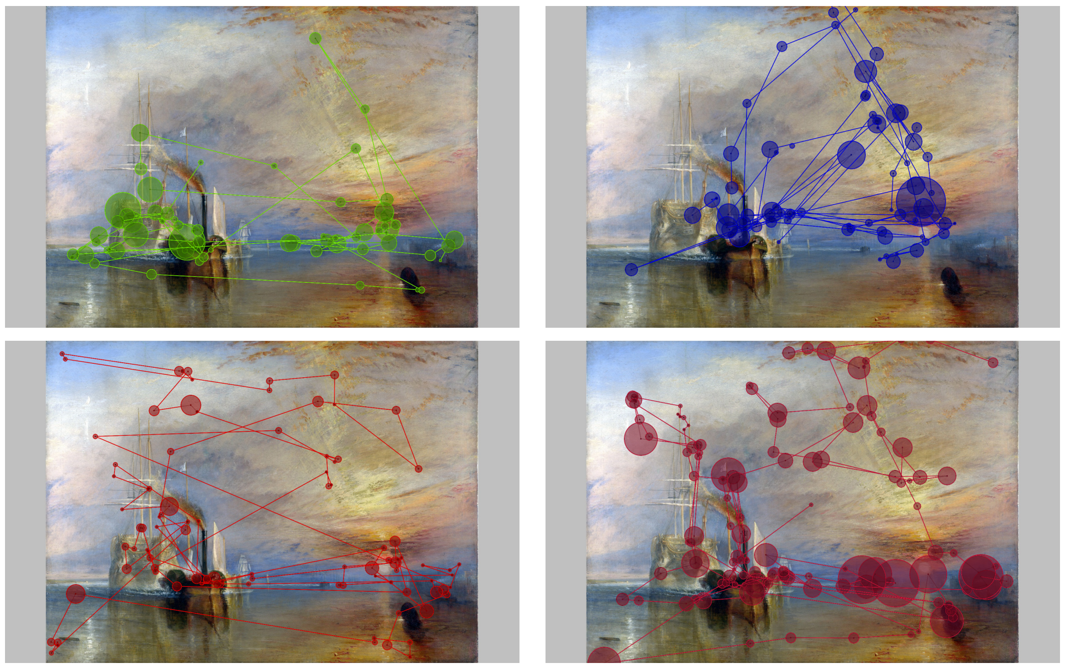

Figure 4.1: From left to right, we show example scanpaths from one subject of each group. The first row represents the scanpath of an expert, instructed and non-instructed. Similarly, the second row represents the scanpath of a novice, instructed and non-instructed.

Figure 4.1 consists of sample images from our experiment. The size of the circles connecting the scanpaths depicts the amount of time a subject fixated in that particular location.

As part of our analysis, we used two MFPA algorithms explained in [23]. The first one is the summary statistics method. The second one uses the spatial characteristics of the scanpath and is based on Gaussian Mixture Models and Fisher vectors. The output obtained from these methods are then fed into a classifier for the classification of our subjects. These MFPA methods were then modified and extended to use a deep Convolutional Neural Network (CNN), and a comparison was made to see how MFPA and the extended MFPA performed on our data.

In the following sections, we start with a detailed explanation of algorithms we used from [23] followed by our approach to extend those algorithms.

## 4.1 The Summary Statistics Method

The summary statistics (SS) from [23] method acts as a baseline for our method of analysis. The input to this method is the mean number of fixations and fixation duration. Considering a trial with T fixations, we can mathematically represent its information as;

$$X_{trial} = \begin{bmatrix} x_1 & x_2 & \cdots & x_T \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_T \\ y_1 & y_2 & \cdots & y_T \\ d_1 & d_2 & \cdots & d_T \end{bmatrix}. \tag{4.1}$$

The columns in the matrix represent the fixation coordinates and duration. The summary statistics method converts the mean number of fixations and durations into a 2-dimensional feature vector which is represented as

$$\phi_{ss}(X_{trial}) = \begin{bmatrix} T & \frac{1}{T}\sum_{t=1}^{T} d_t \end{bmatrix}^T. \tag{4.2}$$

These feature vectors are for each subject viewing each painting which are then fed into a classifier for classifying our subjects.

## 4.2 Method based on Gaussian Mixture Model and Fisher Vectors

In machine learning, most classifiers require all input feature vectors to have a fixed-dimensionality. However, the number of fixations obtained from each trial varies. The reason behind turning a trial's fixation features into Fisher Vectors is to overcome this challenge and to enable MFPA algorithms to capture diagnostic information.

We implemented an algorithm as described in [23] where the fixation locations (x,y) were converted to Fisher Vectors. An image representation obtained by pooling of local image features is called the Fisher Vector. In problems related to classification, Fisher Vectors are usually used as global image descriptors. Fisher Vector encoding is done by fitting a Gaussian Mixture Model (GMM). $p(X|\Theta)$ is the parametric generative model trained for Fisher Vectors, where X is the training data and $\Theta$ are the model's parameters. The dimension of this representation is $\Theta$ dependent and it changes with the change in the number of parameters in $\Theta$.

We used Scikit Learn to fit a GMM and compute Fisher Vectors [42]. A GMM is said to be composed of K Gaussians, which is mathematically represented as

$$p(x) = \sum_{k=1}^{K} w_k \mathcal{N}(x|\mu_k, \Sigma_k). \tag{4.3}$$

Here $w_k$ are the mixture weights and $\mathcal{N}(x|\mu_k, \Sigma_k)$ are the densities of Gaussians in a mixture, which have mean ($\mu_k$) and diagonal covariance ($\Sigma_k$) of their own. Means and covariances of Gaussians are used to construct Fisher Vectors. These $\mu_k$ and $v_k$ for each k mixture component is given by

$$\mu_k = \frac{1}{T\sqrt{w_k}} \sum_{t=1}^{T} q_{tk} \Sigma_k^{-1}(x_t - \mu_k) \tag{4.4}$$

and

$$v_k = \frac{1}{T\sqrt{2w_k}} \sum_{t=1}^{T} q_{tk} [(\Sigma_k^{-1}(x_t - \mu_k)) \circ (\Sigma_k^{-1}(x_t - \mu_k)) - 1] \tag{4.5}$$

respectively, where T represents the total number of fixations in a trial, $\circ$ is the Hadamard product, 1 is a vector of ones, and $q_{tk}$ is the assignment strength given to each mixture component's fixation features $x_t$ by a GMM. These $\mu_k$ and $v_k$ vectors are then concatenated to create an unnormalized GMM Fisher Vector which is given by the equation,

$$\phi_{GMM}(X_{trial}) = \left[u_1 \quad , v_1, \quad \cdots, \quad u_k, \quad v_k\right]^T. \tag{4.6}$$

While doing this, the number of clusters for GMM were chosen between 1 and 5. Finally, before feeding the Fisher Vectors as an input to the classifier, it is necessary to do the two-step normalization process. This normalization process is critical to achieving state-of-the-art performance for classification as stated in [43]. The two steps are: (I) Sign-preserving square root of the features taken i.e., $f(z) = sign(z)\sqrt{|z\ |}$ and is applied to the features element-wise. (II) The features are then converted to unit length by dividing with their Euclidean norm. The dimensionality of the Fisher vectors is then reduced by using Principal Component Analysis (PCA).

## 4.3 Methods Combining MFPA with a Deep CNN

We modified the original MFPA algorithm from [23] to use visual features extracted using a deep CNN. The idea behind using a CNN with MFPA is to see if there are any differences in the viewing behavior of our subjects when image features surrounding fixated regions are taken into consideration. We use ResNet, a type of CNN for feature extraction [44]. In the next sections, we briefly discuss ResNet followed by our two proposed approaches for feature extraction.

#### ResNet

Since the development of AlexNet, the state-of-the-art CNN architecture is going deeper and deeper. With only five convolutional layers in AlexNet to 16 and 19 layers in VGG, Resnet was designed to go as deep as 50, 100 and 150 layers. It is a known fact that an off-the-shelf CNN model can be used either as a classifier or as a feature extractor. We use a ResNet-50 layer neural network [45] pre-trained on ImageNet dataset [44]. Since this is used for feature extraction, we remove the top dense layers and extract features from a layer of our choice. Dense layers are classic fully-connected layers in a neural network which are removed when using a pre-trained neural network as a feature extractor.

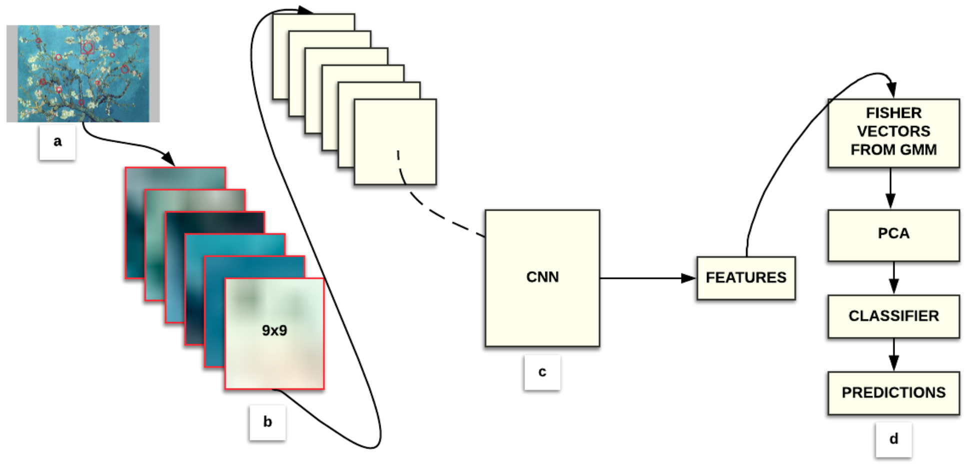

Figure 4.2: Schematic representation of our method. (a) Original image with some fixations. (b) $n \times n$ image patches extracted from areas surrounding each fixation. (c) 50 layer deep convolutional neural network. (d) Flowchart of the modified MFPA algorithm based on Fisher Vectors extracted from GMM.

### 4.3.1 Feature Extraction From Image Patches

Figure 4.2 shows a schematic representation of our first approach in extending MFPA and Figure 4.3 is a flowchart representing the methodology behind our analysis.

As it can be seen, we extract image patches of varied sizes surrounding the fixation locations (x,y). The image patch sizes we consider are 9x9, 17x17, 34x34, and 68x68 in pixels. These sizes correspond to 0.5, 1, 2, and 4-degree visual angle respectively. We do this to see if there are any differences in viewing behavior of our subjects when we alter the algorithm to consider the visual angle.

#### What is a Visual Angle?

Whether a visual stimulus is viewed by a human or a monkey in the laboratory, the size of this visual stimulus is critical. In the studies related to human vision, there is a crucial distinction between the size of an object in the real world and its size when projected onto the retina. The size of an object in the real world can be measured in millimeters, inches,

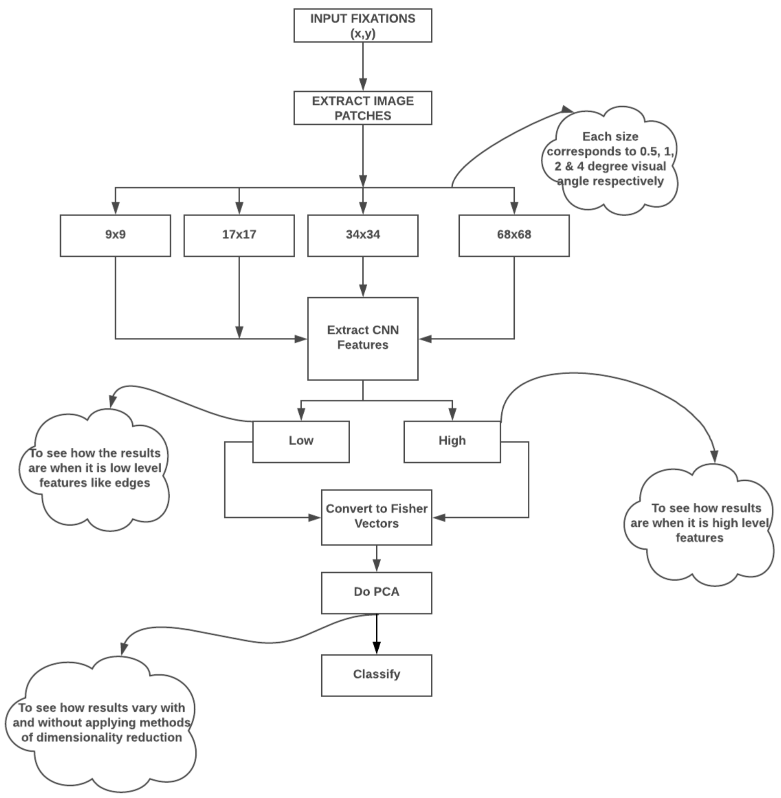
INPUT FIXATIONS (x,y)
EXTRACT IMAGE PATCHES
Each size corresponds to 0.5, 1, 2 & 4 degree visual angle respectively
9x9
17x17
34x34
68x68
Extract CNN Features
Low
High
To see how the results are when it is low level features like edges
To see how results are when it is high level features
Convert to Fisher Vectors
Do PCA
Classify
To see how results vary with and without applying methods of dimensionality reduction

Figure 4.3: A flowchart representing our methodology.

meters, etc. The size of the same object projected on the retina is measured in degrees of the visual angle, a concept that incorporates both the size of an object and its distance from the observer. A good guideline is that, if we held our thumb at arm's length as seen in the Figure 4.4, it is said that the image of the thumb approximately subtends a 2-degree visual angle on our retina.

The idea behind using patches of different sizes corresponding to different visual angles, is to be able to extract low- and high-level features from the patches. We used the Deep Learning library Keras to extract the CNN image features [46]. The low-level image features were extracted from the first convolutional layer while the high-level features were extracted from the last convolutional layer of the network. We obtained $(m \times n)$ CNN image features

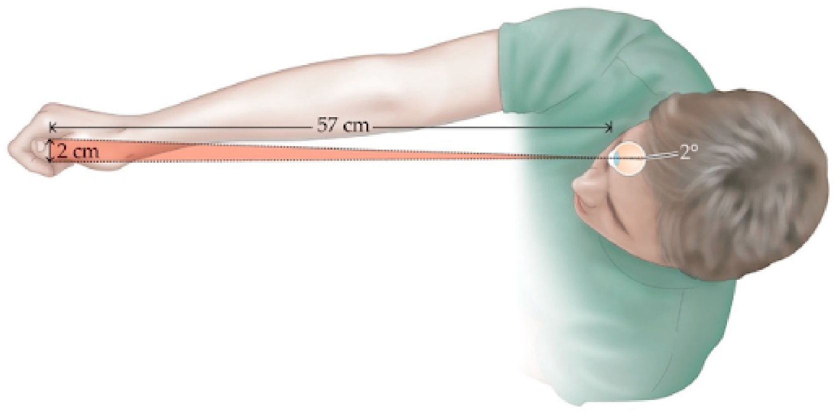

Figure 4.4: Pictorial representation of the concept - Visual Angle.

where m represents the number of image patches, and n represents the flattened dimension of our image features. These features were then converted into Fisher Vectors (FV) before feeding into a classifier.

### 4.3.2 Feature Extraction From Fixation Locations

In our second approach of modifying the original MFPA algorithm, we feed all images in the dataset with fixations to the ResNet50 layer network pre-trained on ImageNet and extract features from the last layer of the network. The reason behind doing this is that we extract features directly from fixation locations. This technique avoids all the features surrounding the fixation locations and helps us understand the features confined to the locations of interest. In our first approach, the patches extracted based on visual angle are fairly small, and this can prevent the CNN from extracting high-level image features as those are associated with semantically meaningful characteristics such as objects and faces which would not be fully contained in the patches extracted. Post feature extraction, we follow the same steps as explained in previous approaches as explained in Section 4.2, i.e. these CNN features are converted to Fisher Vectors and then fed into a classifier.

## 4.4 Classification

Authors from [23] used a Support Vector Machine (SVM) as a classifier. When we used SVM for classification, it happened to be the bottleneck on our dataset. It took approximately 2 hours to process one fold of a cross-validation step when we used SVM as our classifier. So, to optimize computation, we chose to use a Softmax classifier over the SVM.

### 4.4.1 Softmax Classifier

The generalized binary form of logistic regression is a softmax classifier [47]. Just like in hinge loss or squared hinge loss, the mapping function for softmax is defined as the one that takes an input set of data x and maps them to the output class labels with a simple dot product of x and weight matrix W:

$$\mathrm{f}\left(\mathbf{x_i}, \mathbf{W}\right) = \mathbf{W}\mathbf{x_i},$$

But, these scores are interpreted as unnormalized log probabilities for each class label and hence hinge loss is replaced with cross-entropy loss that is defined as,

$$\mathbf{L_i} = -log\frac{\exp\left(\mathbf{f}_{\mathbf{y}_i}\right)}{\sum_j \exp\left(f_j\right)},$$

Where $f_j$ means the j-th element of the vector of class scores f. The mean of $L_i$ overall training examples together with a regularization term R(W) is the full loss for the dataset. The function -

$$\mathrm{softmax}\left(\mathbf{a}\right) = \frac{\exp\left(\mathbf{a}\right)}{\sum_{j=1}^{K} \exp\left(a_j\right)},$$

is called the Softmax function where the vector of arbitrary real valued scores is taken and squashed to a vector of values between zero and one that sum to one. The softmax classifier is hence said to minimize the cross-entropy between the estimated class probabilities and the true distribution.

### 4.4.2 Softmax vs SVM

In addition to the computational efficiency, the advantage behind using a Softmax classifier is that it provides “probabilities” for each class while the SVM computes scores for the classes that are not easy to interpret. For example, given an image, the SVM classifier might give us scores [10.5, 0.6, -21.0] for the classes “bird,” “skateboard,” and “ship.” The softmax classifier can instead compute the probabilities of the three labels as [0.9, 0.09, 0.01], which allows us to interpret its confidence in each class.

We attempted to differentiate novices from experts, instructed from non-instructed subjects and all the four groups. We did 2-way classification to differentiate experts from novices and instructed from non-instructed subjects, whereas 4-way classification was performed to differentiate all four groups. We performed hold-one-out cross-validation for all our conditions to see how expertise and instructions alter the viewing patterns of the subjects.

# Chapter 5

# Results

We compared experts, novices, instructed, and not-instructed subjects, and all the four groups, and reported the classification accuracy calculated from all methods in Tables 5.1, 5.2 and 5.3.

Table 5.1 holds the results obtained from the summary statistics and the original MFPA model. As it can be seen we received slightly above chance results when these methods were implemented on our data.

Table 5.2 holds the results obtained from algorithm-1, i.e. our proposed method involving feature extraction from patches based on the visual angle. Here, MFPA-CNN(L) and MFPA-CNN(H) are the results obtained from our method using low- and high-level CNN image features respectively. The results obtained from this method were better than the original MFPA model but cannot be considered as significantly superior to the earlier approach.

Table 5.3 holds the results obtained from our second proposed method where the entire images were analyzed using CNN and the CNN features were extracted from the coordinates of the fixation locations. As it can be seen, this approach resulted in a significantly better classification accuracy for all our comparisons.

Figures 5.1, 5.2 represent the confusion matrices for all our methods under different conditions, i.e. experts vs. novices, instructed vs. not-instructed and the comparison between all four groups.

Table 5.1: Classification accuracy with their 95% confidence intervals obtained from the original MFPA methods after hold-one-out cross-validation.

| | SS (%) | MFPA (%) |
|---|---|---|
| Exp vs Nov (2-way) | $51 \pm 0.11$ | $55 \pm 0.53$ |
| Inst vs Non-inst (2-way) | $53 \pm 0.26$ | $57 \pm 0.31$ |
| Groups (4-way) | $29 \pm 0.39$ | $30 \pm 0.45$ |

Table 5.2: Classification accuracy with their 95% confidence intervals obtained from our algorithm-1 involving patches of sizes 9x9, 17x17, 34x34, and 68x68 after hold-one-out cross-validation.

| | MFPA-CNN(L) (%) | MFPA-CNN(H) (%) |
|---|---|---|
| Exp vs Nov (2-way) | $58 \pm 0.16$ | $56 \pm 0.39$ |
| Inst vs Non-inst (2-way) | $59 \pm 0.23$ | $58 \pm 0.23$ |
| Groups (4-way) | $30 \pm 0.31$ | $30 \pm 0.57$ |

Table 5.3: Classification accuracy with their 95% confidence intervals obtained from our algorithm-2 after hold-one-out cross-validation.

| | MFPA-CNN(No patches) (%) |
|---|---|
| Exp vs Nov (2-way) | $67 \pm 0.93$ |
| Inst vs Non-inst (2-way) | $79.5 \pm 0.85$ |
| Groups (4-way) | $36.7 \pm 0.96$ |

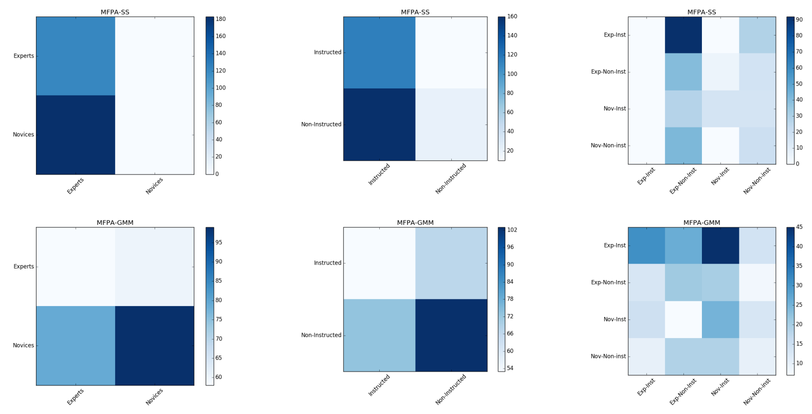

Figure 5.1: In the figure, the first column represents the confusion matrices for comparison between experts and novices, the second column for instructed and non-instructed subjects and the last column represents confusion matrices for all our four groups. Rows represent the methods summary statistics and the original MFPA, respectively. Columns of each confusion matrix represent the predicted classes while the rows represent the ground-truth classes.

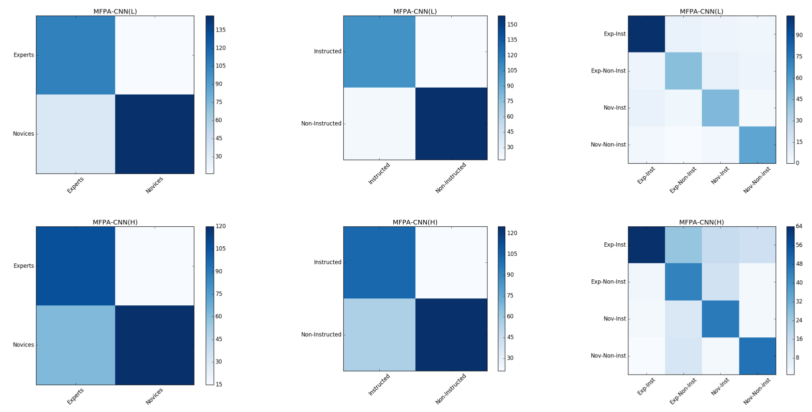

Figure 5.2: In the figure, the first column represents the confusion matrices for comparison between experts and novices, the second column for instructed and non-instructed subjects and the last column represents confusion matrices for all our four groups. Rows represent the methods MFPA-CNN(L) and MFPA-CNN(H), respectively. Columns of each confusion matrix represent the predicted classes while the rows represent the ground-truth classes.

Confusion matrices allow us to evaluate the quality of the output of classifiers. The diagonal elements represent the number of instances for which the predicted label is equal to the true label, while off-diagonal elements are those that are misclassified. The higher the diagonal values of the confusion matrix the better the performance of the classifier, expressed in the number of correct predictions. In Figures, the color coding corresponds to the number of correctly classified or misclassified instances. For example, in Figure 5.1, the top matrix in the first column of the figure shows that the novices were almost always misclassified as experts, which is designated by the deep blue color of the block in the first column, second row of the matrix. Experts, on the other hand, were frequently correctly classified, which is expressed by a lighter shade of blue. The second column of this matrix shows that very few participants were classified as novices resulting in substantial number of misclassifications for the novices.

# Chapter 6

# Discussion

The data represented in Table 5.1, 5.2, 5.3 and Figures 5.1,5.2 show that all algorithms produces an above chance classifications based on accuracy and confidence intervals with the varying performance levels. The above chance results for SS and the original MFPA methods (see Table 5.1) show that there are simple temporal differences when we compare experts and novices with and without instructions and between all the four groups, i.e. experts, novices, instructed and not-instructed.

In contrast, Table 5.2 and 5.3 demonstrate a better differentiation between viewers' gaze patterns using our approaches when compared to results obtained using the summary statistics and the original MFPA method. One aspect to notice from all the tables is, irrespective of the method, the results show a better differentiation of the viewing behavior when the comparison is made between the instructed and the non-instructed conditions. However, an improved performance of our methods mean that although the locations only data can differentiate between instructed and non-instructed subjects, image features provide an important information that yields considerably better results for the classification purposes between these two categories of viewing conditions.

We obtained an above chance results under all our conditions for both low- and high-level image features (see Table 5.2). The size of the image patch surrounding the fixation location did not make any effect on the results obtained. The classification accuracy remained constant with all our patch sizes, 9x9, 17x17, 34x34, and 68x68 pixels. We can thus say that the difference in the features extracted based on the visual angle surrounding the fixation locations do not produce a significant effect on the classification. One possible explanation of this result is that the main differentiating features remain the same in all these patches regardless of the size. Also, the results obtained when high-level image features were extracted is slightly lower when compared to those when low-level features were extracted. The reason behind this could be that the image patches are so tiny that they probably do not include any high-level image features such as faces, eyes. and recognizable objects, and the added features from the larger patches introduce noise from the classification standpoint. Again, these results could also mean that instructions have a larger affect on perception when compared to expertise: although slightly, the results obtained under the instructed condition is better than when expertise is taken into consideration.

In [16], the authors hypothesize that if it can be shown that viewing differences exist in seeing low- and high-level image features, it could potentially mean that novices get attracted to low-level and experts to high-level image features. Hence, based on our results, and discussions from [16, 21], we can argue that novices data contributed to the significant increase in classification accuracy obtained when low-level features were used, and experts' when high-level features were used. This result favors the explanation that the gaze-patterns of novice participants are influenced relatively more by bottom-up factors and gaze patterns of experts are affected more by top-down factors. We can thus support [16, 21] and hypothesize that novices tend to get attracted more towards low-level image features such as color, edges, etc. whereas experts get attracted to high-level features, i.e. areas of an image with more semantically relevant information.

Although, we can hypothesize and support the findings of previously conducted research,our results are not yet sufficient to concretely say how viewing behavior is affected because expertise or instructions. Despite achieving better accuracy than the original model, more work is needed to interpret the results in terms of specific visual characteristics.

Comparing our two methods with and without patch analysis, a substantially better result was obtained for all conditions when we extracted features directly from the fixation locations (see Table 5.3). In this approach, the features are confined to the fixation locations and therefore, do not include the surrounding information. The advantage of this method over the patch-based approach is in that it reduces confusions that can be caused by two fixations very close to each other. For example, in the case of two subjects who view an image and have some fixations very close to each other, patch extraction will include surrounding features of the location that could overlap. In this case, there is a high chance for the algorithm to get confused. However, this will not be the case with extracting features directly from the fixation location. This is potentially the reason behind the significantly better results we obtained from the method involving feature extraction directly from fixation locations of the image.

To gain more insight about our proposed methods, we computed confusion matrices. The confusions indicate that the subjects follow or have more or less similar and generic viewing patterns while viewing paintings. Figure 5.1 represents the confusion matrices obtained from the summary statistics and the original MFPA method and Figure 5.2 represent confusion matrices obtained from our approach where we extracted features from patches surrounding fixation locations. Features that are confused by the original MFPA methods from [23] and our method indicate that the viewing patterns of these subjects are more similar than others. It can be seen that our method significantly reduced the confusions between the experts and novices when compared to those caused by methods implemented from [23], allowing better classification.

Our results also confirm that not just the expertise alone but instructions as well affect viewing behavior of the subjects. Moreover, our findings show that image features obtained from the fixation locations significantly influence the classification accuracy and are more critical than the fixation locations themselves. However, based on our approach of extracting features based on the visual angle, we could only show that the differences exist in the viewing patterns of novices and experts associated with viewing low- and high-level features of images of paintings but could not determine what features the subjects were focusing on. We demonstrate that our method based on the feature extraction directly from the locations of interest is significantly better and reliable over the other proposed methods.

# Chapter 7

# Summary

## 7.1 Conclusions

This thesis discusses the early research and recent advances in visual perception of art by experts and novices and how their eye movements differ. We conducted experiments to determine if there are any differences in gaze patterns of experts and novices viewing images of paintings using machine learning techniques. We proposed two methods that modified the existing Multi-Fixation Pattern Analysis algorithm to include visual features extracted using a Convolutional Neural Network. Both proposed methods resulted in improved classification accuracy. The second method involving feature extraction directly from the fixation locations achieved significantly better performance in comparison to the first method where the features were extracted from image patches. Based on the results the proposed methods can be considered as superior algorithms to the original MFPA algorithms [23]. We can conclude from our findings that image features corresponding to fixation locations provide more meaningful information about the subjects than the locations themselves.

Our results demonstrate that expertise in the field of art and instructions have significant effects on the gaze patterns of viewers. We support the recent work done by [22, 21] and confirm that there is definitely a different viewing pattern that the expert and novice subjects follow while viewing paintings. We successfully differentiated experts from novices when both low- and high-level CNN image features were used and when features were extracted directly from the fixation locations. The influence of instruction was found to have a larger effect on subjects' viewing behavior in comparison to the influence of expertise as evident from our classification results.

## 7.2 Limitations

One important limitation is that the patch sizes we used were too small to capture meaningful high-level features. This could explain the lower classification accuracy obtained for high-level features from the CNN.

As another limitation, we could not determine if saliency plays any role in the differences observed in our experiment because our algorithms were not designed to address saliency. However, it is possible to explore this area with better machine learning models involving neural networks.

## 7.3 Applications and Future Work

In the past, researchers have used algorithms similar to the summary statistics method successfully to predict Schizophrenia, Attention Deficit Hyper Activity Disorder, and Fetal Alcohol Spectrum Disorder [38, 39]. Our extended MFPA algorithms could provide an improvement in screening for these conditions. The early MFPA algorithms could also find a person's identity given a task, our proposed methods in combination with the original MFPA can be used to infer other traits as well. For example, our extended MFPA algorithms could be used to decode the mental state of the subjects.

The future work can further investigate on what image features different groups of observers concentrate when they are viewing art images. One potential approach could use an encoder-decoder network trained on a dataset with fixations as proposed by [36] to predict saliency and visualize and interpret the meaning of the CNN features. This will allow to identify whose vision is more driven towards specific image characteristics and attributes. We believe, our work has wide ranging implications for diverse fields such as imaging and art education in multiple learning contexts, and could also be of particular value to museum curation.

# Bibliography

[1] iMotions. Eye tracker visualizations. *URL:https://imotions.com/blog/7-terms-metrics-eye-tracking.*

[2] Guy Thomas Buswell. How people look at pictures: a study of the psychology and perception in art. 1935.

[3] Alfred L Yarbus. Eye movement and vision, trans. b. haigh. *ed: Plenum Press, New York*, 1967.

[4] Marianne DeAngelus and Jeff B Pelz. Top-down control of eye movements: Yarbus revisited. *Visual Cognition*, 17(6-7):790–811, 2009.

[5] Benjamin W Tatler, Nicholas J Wade, Hoi Kwan, John M Findlay, and Boris M Velichkovsky. Yarbus, eye movements, and vision. *i-Perception*, 1(1):7–27, 2010.

[6] Michelle R Greene, Tommy Liu, and Jeremy M Wolfe. Reconsidering yarbus: A failure to predict observers'task from eye movement patterns. *Vision research*, 62:1–8, 2012.

[7] Ali Borji and Laurent Itti. Defending yarbus: Eye movements reveal observers' task. *Journal of vision*, 14(3):29–29, 2014.

[8] K-H Schlingensiepen, FW Campbell, GE Legge, and TD Walker. The importance of eye movements in the analysis of simple patterns. *Vision Research*, 26(7):1111–1117, 1986.

[9] Susana Martinez-Conde, Stephen L Macknik, and David H Hubel. The role of fixational eye movements in visual perception. *Nature reviews neuroscience*, 5(3):229, 2004.

[10] David Brieber, Helmut Leder, and Marcos Nadal. The experience of art in museums: An attempt to dissociate the role of physical context and genuineness. *Empirical Studies of the Arts*, 33(1):95–105, 2015.

[11] Paul Locher. Contemporary experimental aesthetics: State of the art technology. *i-Perception*, 2(7):697–707, 2011.

[12] Matthew Pelowski and Fuminori Akiba. A model of art perception, evaluation and emotion in transformative aesthetic experience. *New Ideas in Psychology*, 29(2):80–97, 2011.

[13] Johan Wagemans. Towards a new kind of experimental psycho-aesthetics? reflections on the parallellepipeda project. *i-Perception*, 2(6):648–678, 2011.

[14] Elina Pihko, Anne Virtanen, Veli-Matti Saarinen, Sebastian Pannasch, Lotta Hirvenkari, Timo Tossavainen, Arto Haapala, and Riitta Hari. Experiencing art: the influence of expertise and painting abstraction level. 2011.

[15] Calvin F Nodine, Paul J Locher, and Elizabeth A Krupinski. The role of formal art training on perception and aesthetic judgment of art compositions. *Leonardo*, 26(3):219–227, 1993.

[16] WH Zangemeister, Keith Sherman, and Lawrence Stark. Evidence for a global scan-path strategy in viewing abstract compared with realistic images. *Neuropsychologia*, 33(8):1009–1025, 1995.

[17] Stine Vogt. Looking at paintings: patterns of eye movements in artistically naïve and sophisticated subjects. *Leonardo*, 32(4):325–325, 1999.

[18] Paul J Locher, Jeffrey K Smith, and Lisa F Smith. The influence of presentation format and viewer training in the visual arts on the perception of pictorial and aesthetic qualities of paintings. *Perception*, 30(4):449–465, 2001.

[19] Stine Vogt and Svein Magnussen. Expertise in pictorial perception: eye-movement patterns and visual memory in artists and laymen. *Perception*, 36(1):91–100, 2007.

[20] Naoko Koide, Takatomi Kubo, Tomohiro Shibata, and Kazushi Ikeda. Visual fixation patterns of artists and novices in abstract painting observations. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*, pages 1–4. IEEE, 2013.

[21] Naoko Koide, Takatomi Kubo, Satoshi Nishida, Tomohiro Shibata, and Kazushi Ikeda. Art expertise reduces influence of visual salience on fixation in viewing abstract-paintings. *PloS one*, 10(2):e0117696, 2015.

[22] Elena Fedorovskaya, Sanjana Kapisthalam, and Yingtong Bu. Gaze patterns in art viewing and their dependencies on expertise and image characteristics. 2017.

[23] Christopher Kanan, Dina NF Bseiso, Nicholas A Ray, Janet H Hsiao, and Garrison W Cottrell. Humans have idiosyncratic and task-specific scanpaths for judging faces. *Vision research*, 108:67–76, 2015.

[24] Keith Rayner. Eye movements in reading and information processing. *Psychological bulletin*, 85(3):618, 1978.

[25] David Noton and Lawrence Stark. Scanpaths in eye movements during pattern perception. *Science*, 171(3968):308–311, 1971.

[26] David Noton and Lawrence Stark. Eye movements and visual perception. *Scientific American*, 1971.

[27] Anne M Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive psychology*, 12(1):97–136, 1980.

[28] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[29] Derrick Parkhurst, Klinton Law, and Ernst Niebur. Modeling the role of salience in the allocation of overt visual attention. *Vision research*, 42(1):107–123, 2002.

[30] Harold L Kundel, Calvin F Nodine, Elizabeth A Krupinski, and Claudia Mello-Thoms. Using gaze-tracking data and mixture distribution analysis to support a holistic model for the detection of cancers on mammograms. *Academic radiology*, 15(7):881–886, 2008.

[31] Merim Bilalić, Robert Langner, Rolf Ulrich, and Wolfgang Grodd. Many faces of expertise: fusiform face area in chess experts and novices. *Journal of Neuroscience*, 31(28):10206–10214, 2011.

[32] Preethi Vaidyanathan, Jeff Pelz, Cecilia Alm, Pengcheng Shi, and Anne Haake. Recurrence quantification analysis reveals eye-movement behavior differences between experts and novices. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 303–306. ACM, 2014.

[33] Herman F Brandt. The psychology of seeing, new york: Philosophical library, 1945.

[34] François Molnar. About the role of visual exploration in aesthetics. In *Advances in intrinsic motivation and aesthetics*, pages 385–413. Springer, 1981.

[35] Junting Pan, Cristian Canton Ferrer, Kevin McGuinness, Noel E O'Connor, Jordi Torres, Elisa Sayrol, and Xavier Giro-i Nieto. Salgan: Visual saliency prediction with generative adversarial networks. *arXiv preprint arXiv:1701.01081*, 2017.

[36] Sen He, Nicolas Pugeault, Yang Mi, and Ali Borji. What catches the eye? visualizing and understanding deep saliency models. *arXiv preprint arXiv:1803.05753*, 2018.

[37] James R Antes and Arlinda F Kristjanson. Discriminating artists from nonartists by their eye-fixation patterns. *Perceptual and motor Skills*, 73(3):893–894, 1991.

[38] Philip J Benson, Sara A Beedie, Elizabeth Shephard, Ina Giegling, Dan Rujescu, and David St Clair. Simple viewing tests can detect eye movement abnormalities that distinguish schizophrenia cases from controls with exceptional accuracy. *Biological psychiatry*, 72(9):716–724, 2012.

[39] Po-He Tseng, Ian GM Cameron, Giovanna Pari, James N Reynolds, Douglas P Munoz, and Laurent Itti. High-throughput classification of clinical populations from natural viewing eye movements. *Journal of neurology*, 260(1):275–284, 2013.

[40] E. J. Crowley and A. Zisserman. In search of art. In *Workshop on Computer Vision for Art Analysis, ECCV*, 2014.

[41] E. J. Crowley and A. Zisserman. The state of the art: Object retrieval in paintings using discriminative regions. In *British Machine Vision Conference*, 2014.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[43] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. *Computer Vision–ECCV 2010*, pages 143–156, 2010.

[44] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[46] François Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 2015.

[47] CS231. Softmax classifier. *URL:http://cs231n.github.io/linear-classify/softmax.*

# Chapter 8

# Appendix

## 8.1 Python Codes

### 8.1.1 Dataset Manager

```
from collections import deque
from functools import partial
import fixationanalyzer as fa
from fixationanalyzer import Participant
from fixationanalyzer import FeatureExtractor
from itertools import chain
from math import floor
import numpy as np
import csv


class DatasetManager(object):
    """
    Must Modify To Verify Dataset Integrity
    Verify that all participants see all paintings

    """
    def __init__(self,
                    fixation_filename,
                    training_fraction=.7,
                    num_chunks=10,
                    classification_type='exp_v_nov',
                    ):
        self.fixation_filename = fixation_filename
        self.training_fraction = training_fraction
        self.num_chunks = num_chunks
        self.classification_type = classification_type


        self.participants =
           self.__process_input(self.fixation_filename,self.num_chunks)
        self.data_chunks, self.label_chunks =
           self.__chunk_dataset(self.participants)
        self.num_rotations = 0


    def get_train_test(self):
        split_index = floor(self.training_fraction *
           self.num_chunks)

        train_participants = chain(*(self.data_chunks[i]
           for i in range(split_index)))
        test_participants = chain(*(self.data_chunks[i] for
           i in range(split_index,self.num_chunks)))

        train_labels = list( chain( *(self.label_chunks[i]
           for i in range(split_index) ) ) )
        test_labels = list( chain( *(self.label_chunks[i]
           for i in range(split_index,self.num_chunks) ) ) )

        train_data = [p.fixation_data for p in
           train_participants]
        test_data = [p.fixation_data for p in
           test_participants]
        return train_data,train_labels, test_data,
           test_labels

    def rotate(self):

        if self.num_rotations > self.num_chunks:
            raise RuntimeError("You␣can't␣rotate␣your␣
               dataset␣more␣times␣({0})␣than␣there␣are␣
               chunks␣
               ({1})!.".format(self.num_rotations,self.num_chunks))
        self.data_chunks.rotate()
        self.label_chunks.rotate()
        self.num_rotations += 1


    def __chunk_dataset(self,participants):
        if self.classification_type == 'exp_v_nov': #exp vs
           nov
            def _assessment_func(participant):
                return participant.expert

        elif self.classification_type == 'inst_v_ninst':
           #inst vs Non
            def _assessment_func(participant):
                return participant.instructed

        elif self.classification_type == 'four': #4-way
            def _assessment_func(participant):
                truth_table = [[0,1],
                                [2,3]]
        return
           truth_table[participant.expert][participant.instructed]

        label_separated = [[],[]] if
           self.classification_type != 'four' else
           [[],[],[],[]]
```

```
for p in participants:
    lbl = _assessment_func(p)
    label_separated[lbl].append(p)

data_chunks = [ [] for i in range(self.num_chunks) ]
label_chunks = [ [] for i in range(self.num_chunks)
   ]

criteria = any( len(l) for l in label_separated )
chunk_index = 0
while criteria:
    slice = []
    slice_labels = []
    for lbl in range( len(label_separated) ):
        if len(label_separated[lbl]) == 0:
            continue

        slice.append(label_separated[lbl].pop(0))
        slice_labels.append(lbl)

    data_chunks[chunk_index].extend(slice)
    label_chunks[chunk_index].extend(slice_labels)

    chunk_index = (chunk_index + 1) %
       self.num_chunks
    criteria = any( len(l) for l in label_separated
       )

#END WHILE LOOP
data_chunks = deque(data_chunks)
label_chunks = deque(label_chunks)

if any(chunk==[] for chunk in data_chunks):
    raise RuntimeError("All␣your␣folds␣are␣not␣
       populated␣with␣data.␣Can␣you␣get␣more␣
       data?.")
```

```
            return data_chunks,label_chunks

        def
           __process_input(self,fixation_filename,num_data_chunks):

            participants = {}
            #parsing the csv file
            with open(fixation_filename,'r') as f:
                reader = csv.reader(f)
                #retreiving the column headers from the csv file
                column_headers = next(reader)
                column_headers =
                   [header.encode('ascii',errors='ignore').decode('utf-8')
                   for header in column_headers]
                for row in reader:
                    #creating a dictionary for each line of data
                    row_dict = dict( zip(column_headers,row) )
                    subject = row_dict['subject-id']
                    #updating list of participant data
                    if subject in participants:
                        participants[subject].append(row_dict)
                    else:
                        participants[subject] = [row_dict]


            participants = [Participant(subject_id,data) for
               subject_id,data in participants.items()]
            return participants
```

### 8.1.2 Neural Net Feature Extractor

```
from importlib import import_module
import cv2
import numpy as np


class FeatureExtractor(object):
    """
```

```
Class to extract features from pretrained neural
   networks
trained on imagenet with max pooling applied.

This class utilizes keras to automatically leverage
hardware resources and retrieve pretrained networks.
available networks are:
                        - xception
                        - vgg16
                        - vgg19
                        - resnet50
                        - inception_v3
                        - inception_resnet_v2
                        - mobilenet
                        - densenet121
                        - densenet169
                        - densenet201
                        - nasnetlarge
                        - nasnetmobile
                        - mobilenetv2

see: https://keras.io/applications/ for more details
kwargs for network instantiation are:
                                    include_top=False,
                                    weights='imagenet',
                                    pooling='avg'


Instantiation Args:
    network_name (str): name of network to extract
       features from
    interpolation (cv2 constant): type of interpolation
       used to
                                    resize images

Example Use Case:
    network = FeatureExtractor('resnet50')
```

```
        lenna_features = network.extract_features('lenna')

    """
    __SUBMODULES = {'xception':
       'keras.applications.xception',
                    'vgg16': 'keras.applications.vgg16',
                    'vgg19': 'keras.applications.vgg19',
                    'resnet50':
                       'keras.applications.resnet50',
                    'inception_v3':
                       'keras.applications.inception_v3',
                    'inception_resnet_v2':
                       'keras.applications.inception_resnet_v2',
                    'mobilenet':
                       'keras.applications.mobilenet',
                    'densenet121':
                       'keras.applications.densenet',
                    'densenet169':
                       'keras.applications.densenet',
                    'densenet201':
                       'keras.applications.densenet',
                    'nasnetlarge':
                       'keras.applications.nasnet',
                    'nasnetmobile':
                       'keras.applications.nasnet',
                    'mobilenetv2':
                       'keras.applications.mobilenetv2',
                    }
    __FUNCTION_NAMES = {'xception': 'Xception',
                        'vgg16': 'VGG16',
                        'vgg19': 'VGG19',
                        'resnet50': 'ResNet50',
                        'inception_v3': 'InceptionV3',
                        'inception_resnet_v2':
                           'InceptionResNetV2',
                        'mobilenet': 'MobileNet',
                            'densenet121': 'DenseNet121',
                            'densenet169': 'DenseNet169',
                            'densenet201': 'DenseNet201',
                            'nasnetlarge': 'NASNetLarge',
                            'nasnetmobile': 'NASNetMobile',
                            'mobilenetv2': 'MobileNetV2',
                            }

    def __init__(self,
                 network_name='inception_v3',
                 pooling_type='avg'):

        self.model, self.preprocess_fn, self.kerasbackend\
            =
               self.__keras_importer(network_name,pooling_type)
        self.network_name = network_name
        self.pooling_type = pooling_type

    def extract_features_from_specific_layer(self, imgs,
       layer_idx):
        """
        Extracts image features from a specific layer the
           neural network specified in
        __init__

        input::
            imgs (list): list of images
            layer_idx: layer from where you want features
        returns::
            features (np.ndarray): features for this image
        """
        imgs = self.__build_image_data(imgs)
        get_features = self.kerasbackend.function
        ([self.model.layers[0].input,
           self.kerasbackend.learning_phase()],
            [self.model.layers[layer_idx].output])
            features = get_features([imgs,0])
batch_features = np.vsplit(features[0],features[0].shape[0])
        features = np.vstack( [arr.reshape(1,arr.size) for
           arr in batch_features] )
        return features


    def extract_features(self, imgs):
        """
        Extracts image features from a the neural network
           specified in
        __init__

        input::
            imgs (list): list of images
        returns::
            features (np.ndarray): features for this image
        """
        # Error checking for img occurs in
           __build_image_data
        imgs = self.__build_image_data(imgs)
        features = self.model.predict(imgs)
        return features

    def __build_image_data(self, imgs):
        """
        this function turns an input numpy array or image
           path into an
        array format which keras requires for network
           feeding
        that format being a 4D tensor
           (batch,rows,cols,bands)
        (batch size will be always be 1 in this case)

        input::
            img (list):
                    list of numpy arrays
            preprocess_fn (func):
                    preprocessing function for image data,
                       output from
                    self.__keras_importer
        returns::
            img_data (np.ndarray):
                    4D numpy array of the form
                       (num_images,rows,cols,bands)
        """
        # the image must be numpy array so it can be
           processed
        if not isinstance(imgs, list):
            raise ValueError("imgs␣must␣be␣a␣list␣of␣numpy␣
               arrays.")

        imgs = np.stack(imgs,axis=0)

        # preprocessing the image
        img_data = self.preprocess_fn(imgs)
        return img_data

    def __keras_importer(self, network_name, pooling_type):
        """
        Retrieves the feature extraction algorithm and
           preprocess_fns
        from keras, only importing the desired model
           specified by the
        network name.

        input::
            network_name (str):
                name of the network being used for feature
                   extraction
            pooling_type (str):
                type of pooling you want to use ('avg' or
                   'max')
        returns::
            1) model (func):
                function that extract features from the NN
            2) preprocess_fn (func):
                function that preprocesses the image for
                   the network
        """
        # checking to make sure network_name is valid
        if network_name not in self.__SUBMODULES:
            error_string = "unknown␣network␣
               '{network_name}',network_name␣must␣be␣one␣of␣
               {network_list}".format(network_name=network_name,
                 network_list=self.__SUBMODULES.keys())
            raise ValueError(error_string)

        assert pooling_type in
           ['avg','max'],"'pooling_type'␣must␣be␣one␣of␣the␣
           following␣strings␣['avg','max']"

        # importing the proper keras model and preprocess_fn
        submodule =
           import_module(self.__SUBMODULES[network_name])
        model_constructor = getattr(submodule,
                                    self.__FUNCTION_NAMES[network_name])
        model = model_constructor(include_top=False,
                                     weights='imagenet',
                                     pooling=pooling_type)

        preprocess_fn = getattr(submodule,
           'preprocess_input')


        from keras import backend as kerasbackend

        return model, preprocess_fn, kerasbackend

    def __del__(self):
            del self.model
            self.kerasbackend.clear_session()



# END
```

### 8.1.3 Fisher Vector Extractor

```
"""
Code from - https://github.com/jonasrothfus
/fishervector/blob/master/fishervector
/FisherVector.py
"""
import numpy as np
from sklearn.mixture import GaussianMixture
import pickle, os

N_Kernel_Choices = [5, 20, 60, 100, 200, 500]

class FisherVectorExtraction(object):
  def __init__(self, n_kernels=1, covariance_type='diag',
     reg_covar=1e-6):
    assert covariance_type in ['diag', 'full']
    assert n_kernels > 0


    self.n_kernels = n_kernels
    self.covariance_type = covariance_type
    self.reg_covar = reg_covar
    self.fitted = False

  def score(self, X):
    return self.gmm.bic(X.reshape(-1, X.shape[-1]))

  def fit(self, X, model_dump_path=None, verbose=True):
    """
```

```
:param X: either a ndarray with 4 dimensions (n_videos,
   n_frames, n_descriptors_per_image, n_dim_descriptor)
          or with 3 dimensions (n_images,
             n_descriptors_per_image, n_dim_descriptor)
:param model_dump_path: (optional) path where the
   fitted model shall be dumped
:param verbose - boolean that controls the verbosity
:return: fitted Fisher vector object
"""

all_feats = []
max_size = max([a.shape[0] for a in X])
num_features = X[0].shape[1]
for i in range(len(X)):
    features_array_init =
       np.zeros([max_size,num_features])
    #import pdb;pdb.set_trace()
    feat = X[i][0]
    features_array_init[0:feat.shape[0],:] = feat
    feats = np.reshape(features_array_init,
    (1
    ,features_array_init.shape[0]
    ,features_array_init.shape[1]))
    all_feats.append(feats)
# import pdb;pdb.set_trace()
X = np.vstack(all_feats)
#import pdb;pdb.set_trace()
#X = np.reshape(X,(1,X.shape[0],X.shape[1]))
if X.ndim == 4:
  self.ndim = 4
  return self._fit(X, model_dump_path=model_dump_path,
     verbose=verbose)

elif X.ndim == 3:
  # import pdb;pdb.set_trace()
  self.ndim = 3
  # import pdb;pdb.set_trace()
```

```
        X = np.reshape(X, [1] + list(X.shape))
        return self._fit(X, model_dump_path=model_dump_path,
           verbose=verbose)

      else:
        # import pdb;pdb.set_trace()
        raise AssertionError("X␣must␣be␣an␣ndarray␣with␣3␣or␣
           4␣dimensions")

    def fit_by_bic(self, X,
       choices_n_kernels=N_Kernel_Choices,
       model_dump_path=None, verbose=True):
      """
      Fits the GMM with various n_kernels and selects the
         model with the lowest BIC
      :param X: either a ndarray with 4 dimensions (n_videos,
         n_frames, n_descriptors_per_image, n_dim_descriptor)
                or with 3 dimensions (n_images,
                   n_descriptors_per_image, n_dim_descriptor)
      :param choices_n_kernels: array of positive integers
         that specify with how many kernels the GMM shall be
         trained
                                default: [20, 60, 100, 200,
                                   500]
      :param model_dump_path: (optional) path where the
         fitted model shall be dumped
      :param verbose - boolean that controls the verbosity
      :return: fitted Fisher vector object
      """
      if X.ndim == 4:
        self.ndim = 4
        return self._fit_by_bic(X,
           choices_n_kernels=choices_n_kernels,
           model_dump_path=model_dump_path, verbose=verbose)

      elif X.ndim == 3:
        self.ndim = 3
        X = np.reshape(X, [1] + list(X.shape))
        return self._fit_by_bic(X,
           choices_n_kernels=choices_n_kernels,
           model_dump_path=model_dump_path, verbose=verbose)

      else:
        raise AssertionError("X␣must␣be␣an␣ndarray␣with␣3␣or␣
           4␣dimensions")

    def predict(self, X, normalized=True):
     """
     Computes Fisher Vectors of provided X
     :param X: either a ndarray with 4 dimensions (n_videos,
        n_frames, n_descriptors_per_image, n_dim_descriptor)
               or with 3 dimensions (n_images,
                  n_descriptors_per_image, n_dim_descriptor)
     :param normalized: boolean that indicated whether
     the fisher vectors shall be normalized -->
     improved fisher vector
      (https://www.robots.ox.ac.uk/~vgg/rg/papers
      /peronnin_etal_ECCV10.pdf)
     :returns fv: fisher vectors
                   if X.ndim is 4 then returns ndarray of
                      shape (n_videos, n_frames, 2*n_kernels,
                      n_feature_dim)
                   if X.ndim is 3 then returns ndarray of
                      shape (n_images, 2*n_kernels,
                      n_feature_dim)
     """
     all_feats = []
     max_size = max([a.shape[0] for a in X])
     num_features = X[0].shape[1]
     for i in range(len(X)):
         features_array_init =
            np.zeros([max_size,num_features])
         feat = X[i][0]
         features_array_init[0:feat.shape[0],:] = feat
```

```
            feats = np.reshape(features_array_init,
            (1
            ,features_array_init.shape[0]
            ,features_array_init.shape[1]))
            all_feats.append(feats)
        #import pdb;pdb.set_trace()
        X = np.vstack(all_feats)

        if X.ndim == 4:
          return self._predict(X, normalized=normalized)

        elif X.ndim == 3:
          orig_shape = X.shape
          X = np.reshape(X, [1] + list(X.shape))
          result = self._predict(X, normalized=normalized)
          #import pdb;pdb.set_trace()
          return np.reshape(result, (orig_shape[0], 2 *
             self.n_kernels, orig_shape[-1]))

        else:
          raise AssertionError


      def _fit(self, X, model_dump_path=None, verbose=True):
        """
        :param X: shape (n_videos, n_frames,
           n_descriptors_per_image, n_dim_descriptor)
        :param model_dump_path: (optional) path where the
           fitted model shall be dumped
        :param verbose - boolean that controls the verbosity
        :return: fitted Fisher vector object
        """
        assert X.ndim == 4
        self.feature_dim = X.shape[-1]

        X = X.reshape(-1, X.shape[-1])
```

```
    # fit GMM and store params of fitted model
    self.gmm = gmm =
       GaussianMixture(n_components=self.n_kernels,
       covariance_type=self.covariance_type,reg_covar=self.reg_covar,
       max_iter=1000).fit(X)
    self.covars = gmm.covariances_
    self.means = gmm.means_
    self.weights = gmm.weights_


    # if cov_type is diagonal - make sure that covars holds
       a diagonal matrix
    if self.covariance_type == 'diag':
      cov_matrices = np.empty(shape=(self.n_kernels,
         self.covars.shape[1], self.covars.shape[1]))
      for i in range(self.n_kernels):
        cov_matrices[i, :, :] = np.diag(self.covars[i, :])
      self.covars = cov_matrices

    assert self.covars.ndim == 3
    self.fitted = True
    if verbose:
      print('fitted␣GMM␣with␣%i␣kernels'%self.n_kernels)

    if model_dump_path:
      with open(model_dump_path, 'wb') as f:
        pickle.dump(self,f, protocol=4)
      if verbose:
        print('Dumped␣fitted␣model␣to', model_dump_path)

    return self

  def _fit_by_bic(self, X,
     choices_n_kernels=N_Kernel_Choices,
     model_dump_path=None, verbose=True):
    """
    Fits the GMM with various n_kernels and selects the
```

```
        model with the lowest BIC
    :param X: shape (n_videos, n_frames,
        n_descriptors_per_image, n_dim_descriptor)
    :param choices_n_kernels: array of positive integers
        that specify with how many kernels the GMM shall be
        trained
                                    default: [20, 60, 100, 200,
                                        500]
    :param model_dump_path: (optional) path where the
        fitted model shall be dumped
    :param verbose - boolean that controls the verbosity
    :return: fitted Fisher vector object
    """

    bic_scores = []
    for n_kernels in choices_n_kernels:
      self.n_kernels = n_kernels
      bic_score = self.fit(X, verbose=False).score(X)
      bic_scores.append(bic_score)

      if verbose:
        print('fitted␣GMM␣with␣%i␣kernels␣-␣BIC␣=␣
            %.4f'%(n_kernels, bic_score))

    best_n_kernels =
        choices_n_kernels[np.argmin(bic_scores)]

    self.n_kernels = best_n_kernels
    if verbose:
      print('Selected␣GMM␣with␣%i␣kernels' % best_n_kernels)

    return self.fit(X, model_dump_path=model_dump_path,
        verbose=True)

  def _predict(self, X, normalized=True):
    """
    Computes Fisher Vectors of provided X
```

```
:param X: features - ndarray of shape (n_videos,
   n_frames, n_features, n_feature_dim)
:param normalized: boolean that indicated whether the
   fisher vectors shall be normalized --> improved
   fisher vector
:returns fv: fisher vectors - ndarray of shape
   (n_videos, n_frames, 2*n_kernels, n_feature_dim)
"""
assert self.fitted, "Model␣(GMM)␣must␣be␣fitted"
assert self.feature_dim == X.shape[-1], "Features␣must␣
   have␣same␣dimensionality␣as␣fitted␣GMM"
assert X.ndim == 4

n_videos, n_frames = X.shape[0], X.shape[1]

X = X.reshape((-1, X.shape[-2], X.shape[-1]))
   #(n_images, n_features, n_feature_dim)
X_matrix = X.reshape(-1, X.shape[-1])

# set equal weights to predict likelihood ratio
self.gmm.weights_ = np.ones(self.n_kernels) /
   self.n_kernels
likelihood_ratio =
   self.gmm.predict_proba(X_matrix).reshape(X.shape[0],
   X.shape[1], self.n_kernels)

var = np.diagonal(self.covars, axis1=1, axis2=2)

norm_dev_from_modes = ((X[:,:, None, :] -
   self.means[None, None, :, :])/ var[None, None, :,
   :]) # (n_images, n_features, n_kernels, n_featur_dim)

# mean deviation
mean_dev = np.multiply(likelihood_ratio[:,:,:, None],
   norm_dev_from_modes).mean(axis=1) #(n_images,
   n_kernels, n_feature_dim)
mean_dev = np.multiply(1 / np.sqrt(self.weights[None,
            :,  None]), mean_dev) #(n_images, n_kernels,
            n_feature_dim)

        # covariance deviation
        cov_dev = np.multiply(likelihood_ratio[:,:,:, None],
            norm_dev_from_modes**2 - 1).mean(axis=1)
        cov_dev = np.multiply(1 / np.sqrt(2 *
            self.weights[None, :,  None]), cov_dev)

        fisher_vectors = np.concatenate([mean_dev, cov_dev],
            axis=1)

    # final reshape - separate frames and videos
        assert fisher_vectors.ndim == 3
        fisher_vectors = fisher_vectors.reshape((n_videos,
            n_frames, fisher_vectors.shape[1],
            fisher_vectors.shape[2]))


        if normalized:
          fisher_vectors = np.sqrt(np.abs(fisher_vectors)) *
              np.sign(fisher_vectors) # power normalization
          fisher_vectors = fisher_vectors /
              np.linalg.norm(fisher_vectors,
              axis=(2,3))[:,:,None,None]

        fisher_vectors[fisher_vectors < 10**-4] = 0

        assert fisher_vectors.ndim == 4

        return fisher_vectors
```

### 8.1.4 Main Calling Script

```
    import cv2
    import os
    import math
    import numpy as np
```

```
import fixationanalyzer as fa
from fixationanalyzer import FeatureExtractor
from fixationanalyzer import FisherVectorExtraction
from fixationanalyzer import SVM_Classifier
from fixationanalyzer import DNNClassifier
from sklearn.decomposition import PCA
from datetime import datetime
import tensorflow as tf
import time


CONFIGS = {}


IMG_CACHE = {}
def get_image_patches(fixations,labels):
    """
    Generator which returns batches of image patches of
       specified size

    """

    patch_batch = []
    #label_batch = []
    batch_index = 1
    for paintings,label in zip(fixations,labels):
        for painting_id,painting_fixations in
           paintings.items():
            #adding the image to image cache if it's not
               already there
            if painting_id not in IMG_CACHE:
                image_name = os.path.join
                (CONFIGS['IMAGE_DIR'],str(painting_id
                )+'.jpg')
                IMG_CACHE[painting_id] =
                   cv2.imread(image_name)
```

```
painting_img = IMG_CACHE[painting_id]
height,width = painting_img.shape[:2]
max_left = math.ceil(CONFIGS['PATCH_LENGTH']/2)
max_right = width -
   math.ceil(CONFIGS['PATCH_LENGTH']/2)
max_top = math.ceil(CONFIGS['PATCH_LENGTH']/2)
max_bottom = height -
   math.ceil(CONFIGS['PATCH_LENGTH']/2)

for i,fix in enumerate(painting_fixations):

    #throwing away fixation data that is close
       to the image border
    X,Y = fix['X'],fix['Y']
    if X<max_left:
        continue
    if X>max_right:
        continue
    if Y<max_top:
        continue
    if Y>max_bottom:
        continue
    left = X - (CONFIGS["PATCH_LENGTH"] // 2)
    right = X + (CONFIGS["PATCH_LENGTH"] -
       CONFIGS["PATCH_LENGTH"]//2)
    top = Y - (CONFIGS["PATCH_LENGTH"] // 2)
    bottom = Y + (CONFIGS["PATCH_LENGTH"] -
       CONFIGS["PATCH_LENGTH"]//2)
    patch = painting_img[
        top:bottom,left:right,:].astype
        (np.float32)
    patch_batch.append( patch )
    #label_batch.append( label )

if len(patch_batch) > 100:
    patch_batch = patch_batch[0:100]
    yield (patch_batch,label)
```

```
            if patch_batch != [] and len(patch_batch) < 100:
                # import pdb;pdb.set_trace()
                yield (patch_batch,label)

            patch_batch = []


def extract_features(fixations,labels):

    patch_gen = get_image_patches(fixations,labels)

    #batches, label_batches =
       get_image_patches(fixations,labels)
    feature_extractor =
       FeatureExtractor(CONFIGS["NETWORK_NAME"],CONFIGS["POOLING_TYPE"])

    all_features = []
    all_labels = []
    # batch_index = 1
    # with tf.device("/gpu:0"):

    for batch,label_batch in patch_gen:
        #import pdb;pdb.set_trace()
        if CONFIGS['ALGORITHM_TYPE'] == 'mfpa-cnn(L)':
            features = feature_extractor
            .extract_features_from_specific_layer
            (batch,CONFIGS["LAYER_IDX"])
        elif CONFIGS['ALGORITHM_TYPE'] == 'mfpa-cnn(H
␣␣␣␣␣␣␣␣)':
        features = feature_extractor.extract_featur
        (batch)
        # batch_index+=1
        # import pdb;pdb.set_trace()
        #features = np.vstack(features)

        #features =
           np.reshape(features,(1,features.shape[0],features.shape[1]))
        all_features.append(features)
        all_labels.append(label_batch)


    fa.info("all␣neural␣net␣features␣belonging␣to␣each␣
       participant␣viewing␣each␣painting␣is␣extracted␣for␣
       algorithm␣type␣
       {0}".format(CONFIGS['ALGORITHM_TYPE']))
    #import pdb;pdb.set_trace()

    #all_features = np.vstack(all_features)
    del feature_extractor
    return all_features,all_labels


def format_fixations(fixations):
    pass

def test(configs):
    global CONFIGS
    CONFIGS = configs
    CONFIGS['FIXATION_FILENAME'] =
       str(CONFIGS['FIXATION_FILENAME'])
    CONFIGS['TRAINING_FRACTION'] =
       float(CONFIGS['TRAINING_FRACTION'])
    CONFIGS['K_FOLDS'] = int(CONFIGS['K_FOLDS'])
    CONFIGS['CLASSIFICATION_TYPE'] =
       str(CONFIGS['CLASSIFICATION_TYPE'])
    CONFIGS['IMAGE_DIR'] = str(CONFIGS['IMAGE_DIR'])
    CONFIGS['ALGORITHM_TYPE'] =
       str(CONFIGS['ALGORITHM_TYPE'])
    CONFIGS['PATCH_LENGTH'] = int(CONFIGS['PATCH_LENGTH'])
    CONFIGS['BATCH_SIZE'] = int(CONFIGS['BATCH_SIZE'])
    CONFIGS['NETWORK_NAME'] = str(CONFIGS['NETWORK_NAME'])
    CONFIGS['POOLING_TYPE'] = str(CONFIGS['POOLING_TYPE'])
    CONFIGS['LAYER_IDX'] = int(CONFIGS['LAYER_IDX'])
```

```
CONFIGS['N_KERNELS'] = int(CONFIGS['N_KERNELS'])
CONFIGS['COVARIANCE_TYPE'] =
   str(CONFIGS['COVARIANCE_TYPE'])
CONFIGS['REG_COVAR'] = float(CONFIGS['REG_COVAR'])
CONFIGS['N_COMPONENTS_FOR_PCA'] =
   int(CONFIGS['N_COMPONENTS_FOR_PCA'])
CONFIGS['KERNEL_TYPE'] = str(CONFIGS['KERNEL_TYPE'])
CONFIGS['C'] = float(CONFIGS['C'])
CONFIGS['LEARNING_RATE'] =
   float(CONFIGS['LEARNING_RATE'])


start_datetime = datetime.now()
dataset_manager =
   fa.DatasetManager(CONFIGS["FIXATION_FILENAME"],
    training_fractio=CONFIGS["TRAINING_FRACTION",
    num_chunks=CONFIGS["K_FOLDS"],
    classification_type
    =CONFIGS["CLASSIFICATION_TYPE"])

accuracies_k_folds = []



for k in range(CONFIGS["K_FOLDS"]):
    k += 1

    startTime = time.time()
    train_fixations,train_labels,test_fixations,test_labels
       = dataset_manager.get_train_test()
    stopTime = time.time()
    print ("Time:␣" + str(stopTime - startTime))

    if CONFIGS["ALGORITHM_TYPE"] in
       ['mfpa-cnn(L)','mfpa-cnn(H)']:

        #generate features on data
```

```
            startTime = time.time()
            train_features,train_labels =
               extract_features(train_fixations,train_labels)
            #train_features =
               np.expand_dims(train_features,axis=1)
            test_features,test_labels =
               extract_features(test_fixations,test_labels)
            #test_features =
               np.expand_dims(test_features,axis=1)
            stopTime = time.time()
            print ("Time:␣" + str(stopTime - startTime))
            startTime = time.time()
            fisher_vector_extraction =
               FisherVectorExtraction(CONFIGS['N_KERNELS'],
               CONFIGS['COVARIANCE_TYPE'],
               CONFIGS['REG_COVAR'])
            fa.info("Fitting␣GMM␣on␣train␣set.␣",'
␣␣␣␣␣␣␣␣␣␣␣␣(fold={},permutation={})'.format(k
            ,CONFIGS['PERM_IDX']))
            fisher_vector_extraction.fit
            (train_features)
            stopTime = time.time()
            print ("Time:␣" + str(stopTime -
            startTime))
            fa.info("Predicting␣Fisher␣Vectors␣on
␣␣␣␣␣␣␣␣␣␣␣␣Train␣set.␣",'(fold={},permutation={})'
            .format(k,CONFIGS['PERM_IDX']))
            train_fisher_vectors =
            fisher_vector_extraction.predict
            (train_features)
            train_fisher_vectors = np.reshape
            (train_fisher_vectors
            ,(train_fisher_vectors.shape[0]
            ,train_fisher_vectors.shape[1]
            *train_fisher_vectors.shape[2]))
            fa.info("Predicting␣Fisher␣Vectors␣on
␣␣␣␣␣␣␣␣␣␣␣␣Test␣set.␣",'(fold={},permutation={})'
```

```
.format(k,CONFIGS['PERM_IDX']))
test_fisher_vectors =
fisher_vector_extraction.predict
(test_features)
test_fisher_vectors = np.reshape
(test_fisher_vectors,(test_fisher_vectors
.shape[0],test_fisher_vectors.shape[1]
*test_fisher_vectors.shape[2]))

#Reducing dimensions with pCA
pca =
   PCA(n_components=CONFIGS['N_COMPONENTS_FOR_PCA'])
pca.fit(train_fisher_vectors)
train_fisher_vectors =
   pca.transform(train_fisher_vectors)
test_fisher_vectors =
   pca.transform(test_fisher_vectors)
#Classification
fa.info("Entering␣classification␣mode.␣"
,'(fold={}/{},permutation={}/{})'.format
(k,CONFIGS['K_FOLDS'],CONFIGS['PERM_IDX']
,CONFIGS['NUM_PERM']))


if CONFIGS['CLASSIFICATION_TYPE'] == 'four':
    num_classes = 4
else:
    num_classes = 2
#import pdb;pdb.set_trace()
startTime = time.time()
classifier = DNNClassifier
(train_fisher_vectors,
train_labels,
test_fisher_vectors,
test_labels,
CONFIGS['LEARNING_RATE'],
num_classes,
            CONFIGS['BATCH_SIZE'],
            loss='sparse_categorical_crossentropy',
            decay = 1e-6,
            momentum = 0.9,
            nesterov = True,
            activation_init = 'relu',
            activation_final = 'softmax',
            dropout = 0.5,
            n_epochs = 10)
            #import pdb;pdb.set_trace()
            accuracy = classifier.accuracy
            accuracies_k_folds.append(accuracy)
            stopTime = time.time()
            print ("Time:␣" + str(stopTime - startTime))
        #
        dataset_manager.rotate()


k_fold_classification_accuracy_with_mean =
   np.mean(accuracies_k_folds)
k_fold_classification_accuracy_with_std =
   np.std(accuracies_k_folds)*2

print("classification␣accuracy␣{0}%␣for␣algorithm_type␣
   {1}␣with␣95%␣CI␣
   {2}.".format(k_fold_classification_accuracy_with_mean*100,



with open('results_{}.txt'.format(start_datetime),'w')
   as f:
    f.write("START_DATETIME:␣{}".format(start_datetime))
    f.write("\n")
    f.write("FIXATION_FILENAME:␣
       {}".format(CONFIGS['FIXATION_FILENAME']))
    f.write("\n")
    f.write("TRAINING_FRACTION:␣
       {}".format(CONFIGS['TRAINING_FRACTION']))
    f.write("\n")
    f.write("K_FOLDS:␣{}".format(CONFIGS['K_FOLDS']))
    f.write("\n")
    f.write("CLASSIFICATION_TYPE:␣
       {}".format(CONFIGS['CLASSIFICATION_TYPE']))
    f.write("\n")
    f.write("IMAGE_DIR:␣
       {}".format(CONFIGS['IMAGE_DIR']))
    f.write("\n")
    f.write("ALGORITHM_TYPE:␣
       {}".format(CONFIGS['ALGORITHM_TYPE']))
    f.write("\n")
    f.write("PATCH_LENGTH:␣
       {}".format(CONFIGS['PATCH_LENGTH']))
    f.write("\n")
    f.write("BATCH_SIZE:␣
       {}".format(CONFIGS['BATCH_SIZE']))
    f.write("\n")
    f.write("NETWORK_NAME:␣
       {}".format(CONFIGS['NETWORK_NAME']))
    f.write("\n")
    f.write("POOLING_TYPE:␣
       {}".format(CONFIGS['POOLING_TYPE']))
    f.write("\n")
    f.write("LAYER_IDX:␣
       {}".format(CONFIGS['LAYER_IDX']))
    f.write("\n")
    f.write("N_KERNELS:␣
       {}".format(CONFIGS['N_KERNELS']))
    f.write("\n")
    f.write("COVARIANCE_TYPE:␣
       {}".format(CONFIGS['COVARIANCE_TYPE']))
    f.write("\n")
    f.write("REG_COVAR:␣
       {}".format(CONFIGS['REG_COVAR']))
    f.write("\n")
```

```
            f.write("N_COMPONENTS_FOR_PCA:␣
               {}".format(CONFIGS['N_COMPONENTS_FOR_PCA']))
            f.write("\n")
            f.write("KERNEL_TYPE:␣
               {}".format(CONFIGS['KERNEL_TYPE']))
            f.write("\n")
            f.write("C:␣{}".format(CONFIGS['C']))
            f.write("\n")
            f.write("ACCURACY:{}".format
            (k_fold_classification_accuracy_with_mean))
            f.write("\n")
            f.write("95CI:{}".format
            (k_fold_classification_accuracy_with_std))
            f.write("\n")

        CONFIGS = {}

    if __name__ == "__main__":
        fa.main()
```