An Intelligent Chess Piece Detection Tool

Richardson Menezes[1], **Helton Maia**[2]

[1] Department of Computer Engineering and Automation
UFRN, Natal-RN, Brazil

[2]School of Science and Technology
UFRN, Natal-RN, Brazil

helton.maia@ufrn.br

Abstract. *Chess is one of the most researched domains in the annals of artificial intelligence. The main objective of this research is to develop a platform that can determine piece positioning during chess games. Digital image processing methods and real-time object detection (YOLO version 4) algorithms were used during computational development. The problem entails analyzing images captured during a chess game and determining the location of each square on the board, as well as the position of each piece in play. This procedure is repeated at each game turn, enabling the developed system to save and watch all piece moves during a game. The obtained results demonstrate the system's reliability and feasibility.*

1. Introduction

Under the umbrella of pattern recognition, computer vision encompasses some fascinating challenges, such as image classification and object identification. In recent years, significant scientific progress has occurred in these areas, primarily due to advances in convolutional neural networks, deep learning techniques, and increased parallel processing power provided by graphics processing units (GPUs).

The image classification problem entails labeling an input image from a predetermined set of categories. Despite its seeming simplicity, this is one of the most challenging problems in computer vision, where many practical applications and tools are developed. Significant advances have been made in classifying skin cancer images [Esteva et al. 2017]. Using high-resolution images to detect natural disasters such as floods, volcanoes, and severe droughts [Jayaraman et al. 1997, Leonard et al. 2014, Kogan 1997]. This technique is frequently used to evaluate experiment data in neuroscience [de Menezes et al. 2018, de Menezes et al. 2020, Menezes et al. 2022]. Or even for art, such as the classification of paintings by famous artists [de Menezes et al. 2021].

Features used to feed specialized image classification algorithms significantly impact their performance [Srinivas et al. 2017]. This means that the advancement of image classification techniques based on machine learning has been heavily reliant on feature selection engineering, which is expected for the images that comprise the database. As a result, obtaining these features has become difficult, increasing the complexity and computational cost of more established approaches. For image classification, feature extraction, algorithm selection, and learning are traditionally two independent steps, which have been greatly developed and improved using support vector machines (SVMs).

The most robust object classification and detection algorithms currently use deep learning architectures with specialized layers to automate feature filtering and extraction. Machine learning algorithms with specific learning processes, such as linear regression, support vector machines, and decision trees, all follow the same basic steps: make a prediction, calculate the value of an error function, and adjust the prediction engine based on the received feedback, a process similar to how humans learn. Deep learning brought a revolutionary approach to the problem, aiming to overcome previous shortcomings by learning data abstraction through a stratified description paradigm based on a non-linear transformation [Pan et al. 2018]. The ability of deep learning to learn feature extraction from large datasets is the advantage that has brought this approach to prominence.

Convolutional neural networks (CNNs) are widely used in deep learning algorithms to provide the ability to learn from feature extraction. Convolution is a specialized linear operation that can be thought of as applying a filter to a given input in this context. By adjusting the convolutional filter parameters, the repeated application of filters to an input results in a feature map, which can indicate the locations and strength of a feature detected in the input. The network can learn the best parameters for extracting relevant information from the database by adjusting to reduce error. Several deep neural network-based object detectors have been proposed in recent years, [Deng et al. 2013,Kriegeskorte 2015].

This paper aims to create a CNN-based chess piece detection system capable of correctly identifying the location and identity of each chess piece on a chessboard. The system could be used for various purposes, including assisting players with move recommendations, automatic game analysis, and improving the overall experience of an over-the-board chess game.

2. Methods

2.1. Computational Development

The diagram in Figure 1 depicts the steps required to execute the system proposed in this study.

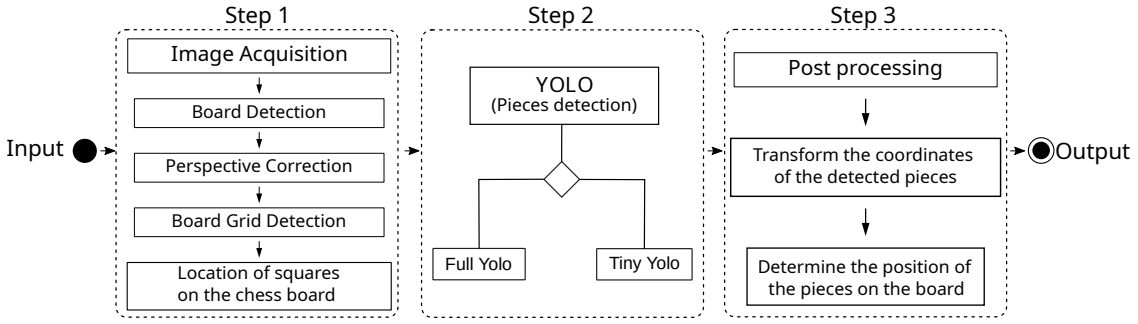

Figure 1. UML activity diagram of the chess piece detection and classification system

Obtaining pictures from a video source, such as a video file or a live camera feed, is the first step in the system's operation, as shown in Figure 1. Traditional image processing techniques are then employed for board localization, perspective correction, board

grid detection, noise filtering, and histogram equalization methods to improve image quality for the following steps. The first step leads to the creation of a mapping of all squares on the board to image space coordinates. To accomplish this, the representation of the squares comprising the board was adopted by two points, the upper left and lower right, to determine whether a new sample point in the image is within the area delimited by a specific square.

The second step's goal is to recognize the pieces on the board. The YOLO network is used to process the original frame captured by the camera, which can take two different paths. The Full YOLO processing path employs the most powerful version of the YOLO algorithm for object detection, resulting in more accurate predictions; however, running this network requires more computational power for network predictions to run in real-time. The object detector's Tiny YOLO architecture is a more simplified version of the network that achieves performance comparable to its more powerful counterpart in some scenarios; thus, this version is ideal for real-time video processing. As a result of step two, the weights from both networks' training are adjusted to detect the chess pieces on the board. In this step, the user selects which version to use based on which best meets their needs.

Finally, the third step applies post-processing operations to the results of the previous steps. Initially, the results of the predicted locations of the detected pieces are combined with mapping the squares on the board. For this purpose, the coordinates of the detections are transformed into image space with a perspective transformation, and then each piece's square is ascertained. As a result, by the end of the third step, the system has a complete mapping of which pieces are in play and in which squares they are located, allowing the creation of a two-dimensional representation of the game's current state.

2.2. Experimental Setup and Dataset

Given the problem addressed by this work, it was necessary to create a customized dataset suitable for detecting chess pieces on a chessboard for the application of the object detection algorithm. Videos of chess games were analyzed to validate the computational development proposed in this work. Figure 2 shows an example of the experimental configuration used to conduct the recordings.

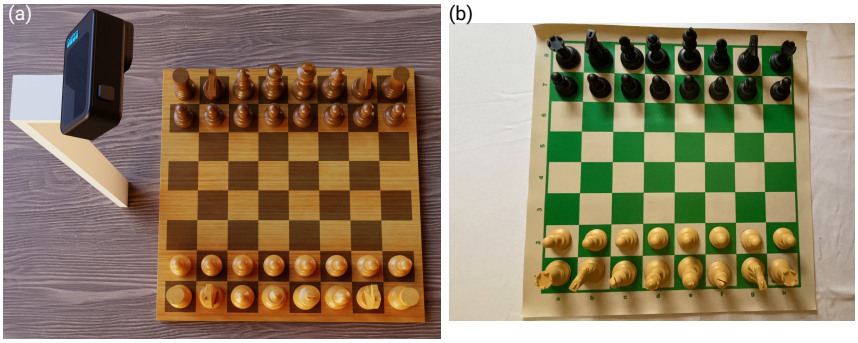

Figure 2. Experimental setup. (a) 3D representation of the experimental setup for recording chess matches, including the camera location for video acquisition; (b) Example frame showing the top view of the chessboard recorded by the camera during the dataset's matches.

This study's experimental configuration was based on images captured with an iPhone 11 smartphone. The camera was set up to capture RGB images at 30 frames per second at a resolution of 1920×1080 pixels.

Figure 2(a) depicts a three-dimensional view of the experimental setup for video acquisition through chess matches. During the recording sessions, the camera was positioned at a height of 61 cm to record games on a board measuring 47 cm x 48 cm, with each square on the board having a side of 5.3 cm.

To validate the developed computational tool, frames were captured in experimental videos of chess matches. Figure 2(b) illustrates the capture made by the proposed experimental configuration of the chessboard ready for the start of a game. The attached camera provides a panoramic view of the board.

Table 1 shows the dataset used to train the object detection algorithm. This comprises, 6317 manually annotated images, with 5054 representing 80% of them being separated to be used exclusively in the object detection algorithm training process and 1263 corresponding to the remaining 20% for the test set, divided into 12 classes representing each chess piece of both colors.

Table 1. Image dataset created for object detection algorithm training and testing

Color	Piece	Training Samples	Test Samples
White	Pawn	28 789	7 234
	Knight	5 881	1 328
	Bishop	4 884	1 144
	Rook	8 254	1 994
	Queen	3 473	883
	King	4 925	1 222
Black	Pawn	28 299	7 000
	Knight	4 902	1 344
	Bishop	5 541	1 219
	Rook	8 799	1 946
	Queen	3 423	874
	King	4 931	1 226
Total		112 101	27 414

3. Results and Discussion

Accurately locating chess pieces on a chessboard is a challenging task that requires modern and efficient computing. The C and Python programming languages and frameworks such as Open-Source Computer Vision (OpenCV) and TensorFlow were used for all algorithmic development in this work.

The application created for this work was ChessPy. A system intended to work best with images in which the chessboard is the biggest object in the scene. The camera can record the complete board, which can be positioned at an angle to the board's plane.

Piece detection aims to locate them in an image and then yield which pieces are present and where they're located. The YOLOv4 [Bochkovskiy et al. 2020] object de-

tection model was selected since it's extremely precise and fast for neural network-based models. This model, after training, can identify objects in real-time, even on lower-end hardware.

To identify chess pieces, two YOLO network setups were used. The results of this research were obtained by analyzing 6317 images arranged according to the data set outlined in Section 2.2.

First, the YOLO Full type network was used to train the object detection model, which uses the Darknet-53 [Redmon and Farhadi 2018] convolutional architecture with 53 convolutional layers. This model was trained from a pre-trained model that used the ImageNet dataset [Russakovsky et al. 2015], using the transfer learning technique described in [Redmon et al. 2015], to fully take advantage of the optimization process and weight adjustment. The model is made up of a segment with convolutional layers and residual connections in a total of $65.29 \cdot 10^9$ floating point operations. Each model requires 235 MB of storage. It took 11.59 hours to train the model.

Afterward, a smaller and faster architectural alternative, the YOLO Tiny network, was also trained. This "tiny" version employs only a portion of Darknet-53's features, 23 convolutional layers, resulting in $9.67 \cdot 10^9$ floating point operations, roughly seven times fewer than its bigger counterpart. Each model requires only 34 MB of storage space. The network was trained following the strategy described in [Redmon et al. 2015] by fine-tuning a pre-trained model using the ImageNet dataset [Russakovsky et al. 2015]. It takes 2.37 hours to train a model. Figure 3 shows some of the results of recognizing chess pieces in the created test dataset.

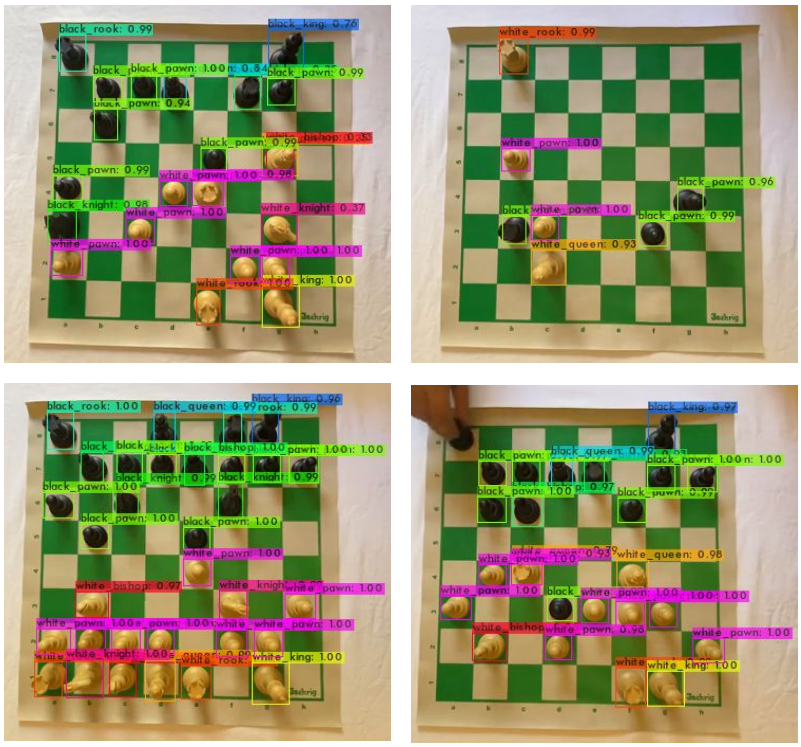

Figure 3. Examples of trained object detection model output

The YOLO architecture's approach to object recognition is faster than other systems in part because it predicts the placement of objects in the image as a whole rather than classifying objects in separate regions, as other networks do [Girshick et al. 2013, Girshick 2015,Ren et al. 2015]. YOLO divides the image into $S \times S$ cells and looks for items within each cell simultaneously. Because only two objects are permitted in the same cell by design, this method has issues coping with close or overlapping objects. However, the algorithm accurately forecasts objects in similar locations, like chess pieces, which are always arranged similarly on the board.

As a single-stage object detector, YOLOv4 is based on the original YOLO model [Redmon et al. 2015] and the YOLOv3 detector [Redmon and Farhadi 2018]. The fourth version is divided into two halves, the first known as the backbone and the second as the head. The backbone is a convolutional network-based feature extractor, whereas the head is the component of the model that predicts object classes and their bounding boxes.

Figure 4 depicts the loss function optimization process for both network designs during the training stage; by the conclusion of the 8000 allotted epochs, this process neared zero. The figure also shows the greater efficiency in optimizing the weights of the Tiny architecture, which achieved smaller values for the loss function at the start of the training procedure and reached values roughly ten times smaller at the end of the optimization process.

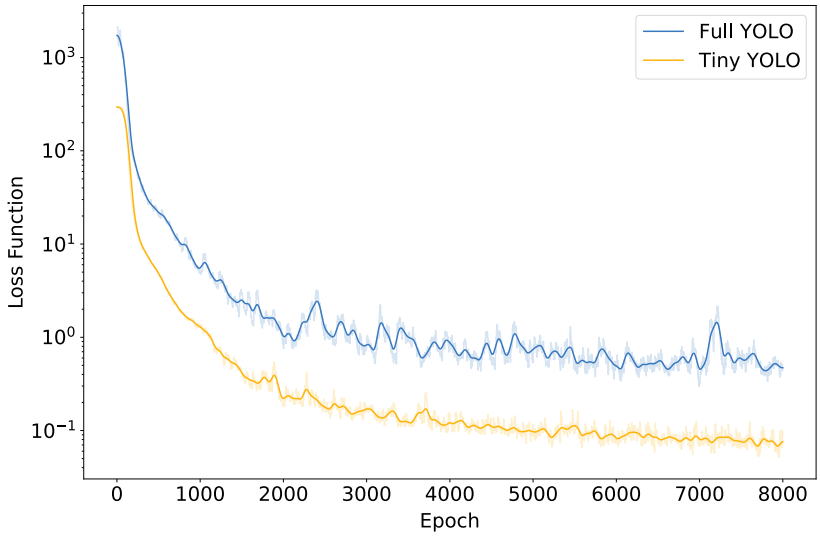

Figure 4. Loss function minimization for the YOLO Full and Tiny architectures during training

Table 2 shows the results of employing the evaluative classification metrics in the model's final step for Full and Tiny architectures. These metrics are used to assess the quality of the classifiers' output. Precision is defined as the ratio of accurately anticipated positive observations to all positive observations. The recall is the ratio of accurately predicted positive observations to all observations. The F1-Score is the weighted average of precision and recall, with a maximum value of 1 signifying flawless precision and recall.

Table 2. Classification metrics on the test dataset for the evaluated models

Model	Precision (%)	Recall (%)	F1-Score
Full YOLO	87	90	0.88
Tiny YOLO	89	89	0.89

Still, in Table 2, the Full network's accuracy, recall, and F1-Score values were all above 80%. Indicating good quality of class predictions and success in the training stage with good generalization, whereas the Tiny network performed slightly better despite its smaller size, with accuracy, recall, and F1-Score equal to 0.89.

Table 3 presents additional results relating to the performance of piece detection. The average precision for each class is displayed first. The classes correspond to each sort of chess piece in both traditional colors, black and white. Finally, at the bottom of the table, the mAP values for each model are presented.

Table 3. Average precision for the detections of each class for the evaluated models

		Average Precision (AP, %) Model	
Color	**Object**	**Full YOLO**	**Tiny YOLO**
White	Pawn	90.88	90.90
	Knight	84.97	84.11
	Bishop	74.09	58.88
	Rook	84.67	84.72
	Queen	72.33	74.28
	King	86.70	84.81
Black	Pawn	90.68	90.89
	Knight	90.13	89.61
	Bishop	67.47	70.10
	Rook	88.18	87.64
	Queen	90.57	89.67
	King	90.84	90.86
mAP		**84.29**	**83.04**

Table 3 shows that the Full network has somewhat higher statistical values for mAP than the Tiny network, with a difference of 1.25%. The Tiny network's reduced size is advantageous here. Despite reducing abstraction capacity, this design outperformed its bigger version in numerous classes. It's also worth mentioning that both algorithms struggled to recognize Bishops in all colors, with the best results of about 70%. This is due to the strong resemblance between Bishops and Pawns, with the latter having more examples of evaluating because their presence on a chessboard is more predominant.

Regarding the intersection over union metric (IoU), the Full architecture achieved a competent generalization for object detection with an average IoU of 71.47%. The Tiny network, the model's smaller counterpart, earned an average value of 77.80% for the IoU metric, demonstrating the model's competency for the stated challenge yet again.

Table 4 shows the ratio of accurate and erroneous detections produced by the network predictions on the test dataset, completing the evaluation of the results for the object detection algorithm. The detector made relatively few errors in predictions, accurately classifying the majority of the test samples, based on its favorable outcomes on the metrics displayed in Table 3.

The difficulty in accurately categorizing the Bishop pieces is evident in the findings provided in Table 4. Tiny architecture was more successful in categorizing such a piece. In the instance of Queens, another challenging piece, the Full network provided the best classification results for both colors of this piece.

Table 4. True and false positives' classification for the evaluated models

		Model			
		Full YOLO		**Tiny YOLO**	
Color	**Object**	**True Positive**	**False Positive**	**True Positive**	**False Positive**
White	Pawn	7 230	719	7 226	846
	Knight	1 059	222	1 065	247
	Bishop	818	462	619	281
	Rook	1 563	115	1 490	17
	Queen	578	270	523	166
	King	1 052	201	1 033	66
Black	Pawn	6 970	609	6 990	480
	Knight	1 121	40	1 190	80
	Bishop	871	821	705	300
	Rook	1 734	379	1 718	512
	Queen	705	5	741	24
	King	1 053	6	1 079	2

Figure 5 shows the efficiency of the Tiny and Full architectures in terms of training and prediction time. Training took about 2.4 hours using the smaller architecture, almost five times faster than the full version. A box plot of the time spent on both designs in classifying a single image is shown in Figure 5(b). The Tiny version's smaller size directly affects its execution time, with mean and standard deviation values of $3.08 \pm 0.05\ ms$ compared to $20.54 \pm 0.09\ ms$ for the Full version.

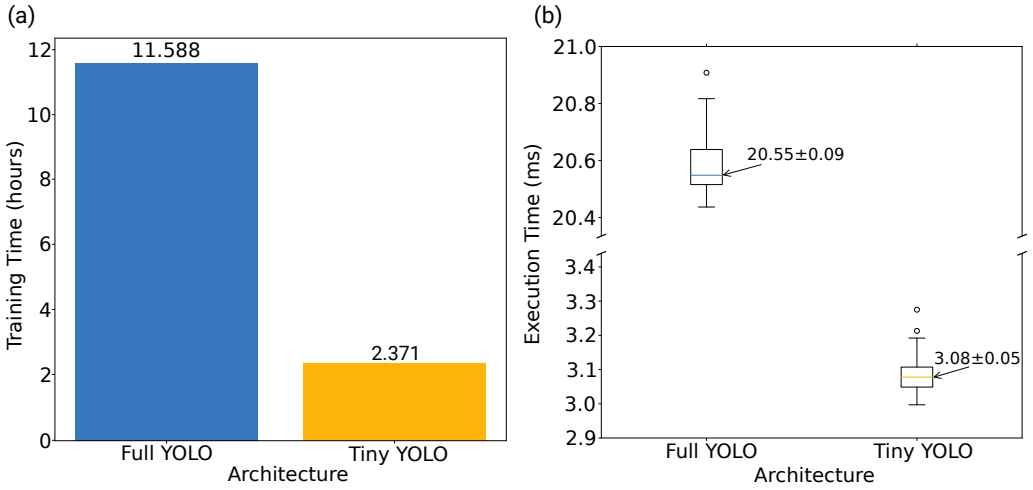

Figure 5. Models' usage of GPU computation time. (a) Time required to train both models for 8000 epochs; (b) Time required to classify a single image

A graphical user interface (GUI), illustrated in Figure 6, was created to facilitate access to the available tools. The user can import the footage, watch the frames being analyzed, and alter various parameters and options in real-time.

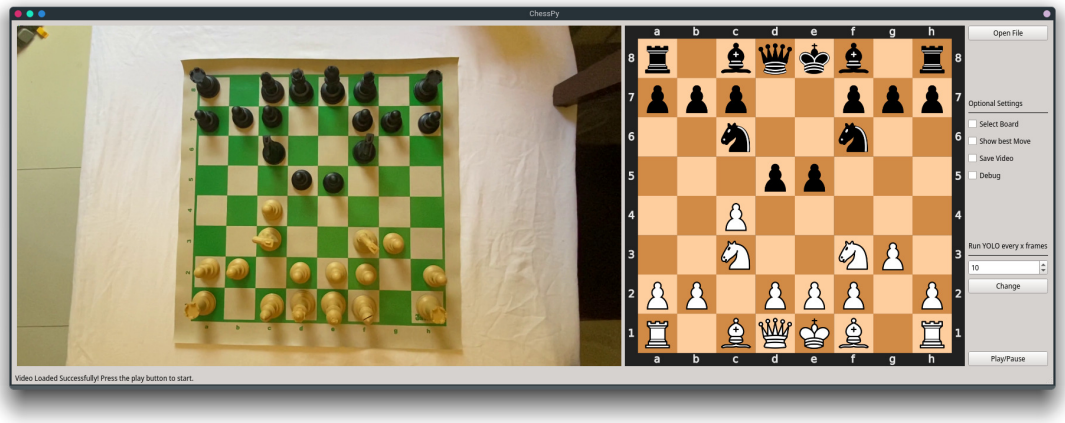

Figure 6. Interface designed for video analysis of a chess match

4. Conclusion

In this paper, a real-time computational tool for monitoring chess matches was designed. The pieces on the board could be located and classified using cutting-edge object recognition methods. Their board position was determined using traditional digital image processing methods.

According to the results discussed in this paper, the presented approach successfully predicts the location of the pieces on the chessboard, with the proposed models attaining accuracy, recall, and F1-Score values reaching 0.89.

The results demonstrate that the system could track objects with high precision, even under challenging settings, with pieces partly obscured by others and shadows projected by the players' arms.

With the solution described here requiring only a low-cost camera, the tool developed in this study allows chess tournament organizers to record and broadcast the event's games in a more accessible and less expensive manner than that provided by sophisticated electronic boards.

Considering the vast variety of chess pieces on the market, creating a dataset with more diverse styles of pieces and boards could be noted as a possible future work suggestion. Thus, the system could immediately serve a wider variety of game scenarios. Furthermore, a more user-friendly UI is being developed so that the games identified by the system can be broadcast live online for tournament monitoring.

References

Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection.

de Menezes, R. S. T., Cordeiro, A. M., Magalhães, R. M., and Maia, H. (2021). Classification of paintings authorship using convolutional neural network. *Sociedade Brasileira de Inteligência Computacional*.

de Menezes, R. S. T., de Azevedo Lima, L., Santana, O., Henriques-Alves, A. M., Santa Cruz, R. M., and Maia, H. (2018). Classification of mice head orientation using support vector machine and histogram of oriented gradients features. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE.

de Menezes, R. S. T., Luiz, J. V. A., Henrique-Alves, A. M., Santa Cruz, R. M., and Maia, H. (2020). Mice tracking using the yolo algorithm. In *Anais do XLVII Seminário Integrado de Software e Hardware*, pages 162–173. SBC.

Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118.

Girshick, R. (2015). Fast r-cnn. arxiv 2015. *arXiv preprint arXiv:1504.08083*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. corr, abs/1311.2524. *arXiv preprint arXiv:1311.2524*.

Jayaraman, V., Chandrasekhar, M., and Rao, U. (1997). Managing the natural disasters from space technology inputs. *Acta Astronautica*, 40(2-8):291–325.

Kogan, F. N. (1997). Global drought watch from space. *Bulletin of the American Meteorological Society*, 78(4):621–636.

Kriegeskorte, N. (2015). Deep neural networks: a new framework for modelling biological vision and brain information processing. *biorxiv*, page 029876.

Leonard, M., Westra, S., Phatak, A., Lambert, M., van den Hurk, B., McInnes, K., Risbey, J., Schuster, S., Jakob, D., and Stafford-Smith, M. (2014). A compound event frame-

work for understanding extreme impacts. *Wiley Interdisciplinary Reviews: Climate Change*, 5(1):113–128.

Menezes, R., de Miranda, A., and Maia, H. (2022). Pymicetracking: An open-source toolbox for real-time behavioral neuroscience experiments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21459–21465.

Pan, W. D., Dong, Y., and Wu, D. (2018). Classification of malaria-infected cells using deep convolutional neural networks. *Machine learning: advanced techniques and emerging applications*, 159.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. arxiv 2015. *arXiv preprint arXiv:1506.02640*.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S., and Babu, R. V. (2017). An introduction to deep convolutional neural nets for computer vision. In *Deep Learning for Medical Image Analysis*, pages 25–52. Elsevier.