Emotion Recognition Using YOLOv3

Abdul Bhutta
Electrical and Computer Engineering
Toronto Metropolitan University (TMU)
Toronto, Canada
abdul.bhutta@torontomu.ca

I. Introduction [Project Proposal]

In 2020, the world faced a pandemic that we have not seen in our generation, forcing many of the countries to be placed in lockdown. This meant many businesses and essential services, including higher education and medical clinics, had to be shut down. Many businesses and services had to adapt and transition to a remote environment, while society was not ready for this occurrence. The medical clinics were all closed, and only hospitals were open; however, they were overrun with COVID-19 patients. Many clinics provided remote services, but it was difficult for doctors to diagnose patients with various symptoms without being able to see them in person. Many students who were suffering from anxiety/stress/depression did not have the right help or support as most campuses were closed. It was challenging for many as the world tried to adapt to the new standards. Doctors could not diagnose patients over the phone or in an online environment or require them to come in person, causing more difficulties for the patient. As a pandemic can occur at any point, we need a system to help doctors aid in a patient's diagnosis. One of the ways a patient can be diagnosed with depression or various conditions is by how they have been feeling, such as sad or anxious.

II. Motivations and Objectives [Project Proposal]

Many parts of the world were unprepared for the COVID-19 pandemic; humanity needs to learn from the past and avoid further pandemics or face severe consequences. As an undergraduate student in 2020, I was faced with a lot of stress and anxiety. Many campus buildings were closed, and family doctors only allowed remote appointments, such as phone calls. Not only was it challenging for patients, but it was difficult for doctors to diagnose patients with diseases or conditions as well. Most on-campus facilities recommended contacting the hospital if your condition worsens, but the wait times were long. Higher education and health care services were not ready to transition to a remote environment, and measures must be taken now for future reference. In Ontario, about 59% of the students felt the pandemic made them more depressed, while 39% made their current mental health worse [1]. We must be ready to face such challenges and require services/applications to help users feel less anxious and aid medical responders with the appropriate diagnosis tools. One of the main objectives for emotion detection in an application can be used by doctors to monitor a patient's mood. The patient can be asked to fill out an online application regarding their conditions and then request them to take a real-time video of themselves to allow the physician to detect their current mood. One of the challenges in this application would come from users falsifying their emotions or answering the questions in a way that allows them to be diagnosed with a condition and access to medical drugs.

III. Related Works

In the course, we covered Multimodal Emotion Recognition using Interpretable Machine learning to detect emotions from vocal and visual expressions. The experiment included 500 videos with eight subjects and six emotions (anger, disgust, fear, happiness, sadness, and surprise), where 360 videos were used for training and 140 videos were used for testing. The experiment involved extracting audio and facial expression features from the videos and using the features to train a model to recognize emotions without the interference of cultural and language background. The visual aspect of the experiment involved extracting facial expression features using key frame extraction, face detection, and Gabor filters. It applied four methods: Gaussian Mixture Model, K-NN, Neural Network, and Fishers Linear Discriminate (FLDA), where FLDA outperformed the other three. This paper compares the initial results from the YOLOv5 model to the Multimodal Emotion Recognition using Interpretable Machine learning to determine if the deep learning models can achieve similar accuracy.

IV. YOLOv5 Algorithm

The YOLOv5 algorithm was selected for this project as it is one of the latest versions of the YOLO family. The other higher versions were not chosen due to the computational resources required and the project's time constraints. Training a model on one of the higher versions would require paying more for compute units on Google Colab, and the training time would increase significantly. The layers increase as you go higher in the YOLO versions, which may lead to overfitting due to the size of the dataset, and the model may not generalize while YOLOv5 has been optimized for speed and accuracy. Object detection is difficult task in computer vision and requires considerable computational power. The emergence of deep learning and cloud computing has allowed the development of object detection algorithms that can detect objects in real-time. The YOLO algorithm is one of the most popular object detection algorithms, with various versions to meet users' needs.

In the context of emotion recognition using the user's facial expressions, it can be considered an object detection problem. Each facial expression has unique key points that can be used to identify the emotion, such as the eyes, mouth, nose, etc. The YOLO algorithm can be used to detect the key points of the face and use them to identify each emotion. Identifying each user's emotion is difficult, but creating a generalized model can be used to determine the emotion to a certain degree. Object detection algorithms have been trained to handle many conditions, such as occlusion, light changes, and real-time processing. In most real-time applications, the lighting conditions are subject to change as the user may be in a dark or bright environment.

Furthermore, the user may be wearing a cap, glasses, or other accessories that may cause occlusion of the face. The YOLO algorithm has been optimized to handle these conditions through deep learning and CNN by involving low-level and high-level features to fuse them to find patterns and complex features in the image. Using the model over time allows it to learn additional features and intricate patterns, leading to better accuracy.

Since each emotion is associated with a facial expression, the assumption is that each facial expression can be expressed as an object. For example, A happy face can be considered an object and the emotion label associated with this "object" is happy. Since there are seven emotions, there will be seven unique "objects." To understand why YOLOv5 is one of the best object detection algorithms, it is essential to understand and analyze the architecture of the YOLOv5 algorithm and how it differs from its predecessor models. This will allow us to understand the improvements made to the YOLOv5 algorithm and how it's been optimized for speed and accuracy. Please note that most of the information about the YOLOv5 algorithm is analyzed and explored through the source code and documentation, as there is no official research paper for YOLOv5. The information about YOLOv3 and YOLOv4 are from the official research papers.

V. YOLO Architecture

YOLO (You Only Look Once) models are object detection models that identify multiple objects within an image from various classes and display bounding boxes to visualize where each object resides. It is a one-stage detection that applies a single pass through a neural network on the entire image. A one-stage detector processes the image only once and predicts the object's class probability and bounding box coordinates at the network output. The neural network creates a grid at three different scales for the current image, with each cell containing the probability of an object, and the cell is at the center of the object.

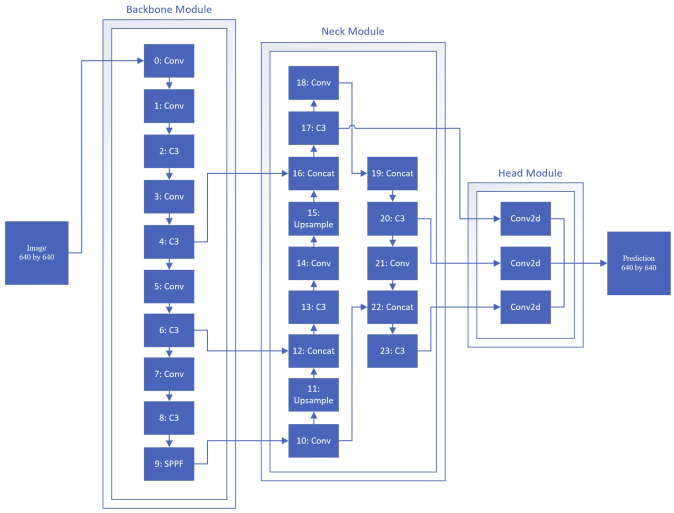


Fig. 1. YOLOv5 Architecture

This section will discuss the comparison between YOLOv5 and its predecessor models while discussing the architecture of YOLOv5 as shown in Figure 1. YOLOv3 and YOLOv4 were created using the Darknet framework and are challenging to implement, while YOLOv5 was created using PyTorch and is much more accessible than the other models. YOLOv1 and YOLOv2 architecture are not discussed in this paper as they are outdated and not used anymore.

The YOLOv3 model consists of a backbone, neck, and head. The backbone is implemented to extract features from the input image using Darknet-53; the extracted features are passed through the neck to refine the features before object detection, and the head is used to predict the labels and bounding boxes. The backbone is used for feature extractions with 53 CNN layers, and another stack of 53 layers is used for detection tasks for a total of 106 layers in the architecture. Each layer in the backbone consists of residual blocks, skip connections, and upsampling. Furthermore, each training batch gets processed through a normalization technique followed by an activation function, Leaky ReLu. The backbone loads the 106 layers with three primary layers in the model used for object detection: 82, 94, and 106 [2].

The YOLOv4 model consists of a CSPDarknet53 backbone, a neck which consists of SPP Block and PANet, and a YOLOv3 head. The backbone has been slightly modified from the Darknet53 backbone to include a CSP (cross-stage partial connections) module to reduce the number of parameters and increase the model's speed. The SPP (Spatial Pyramid Pooling) block is added between the backbone and the PANet to extract features at different scales and sizes. The PANet (Path Aggregation Network) fuses the low-level to high-level features and combines the features from different scales and sizes, leading to better object detection. In YOLOv4, a new activation function called Mish was introduced and implemented, outperforming its previous ReLu and Leaky ReLu functions [4].

The YOLOv5 architecture also consists of a backbone using CSP-Darknet53, a neck, and a head similar to YOLOv3 with various layers in each module as shown in Figure 1. The neck is modified to include a Feature Pyramid Network (FPN) and PAN to extract features at different scales and sizes. The head is similar to the YOLOv3 head, with minor modifications to the detection layers for predicting the bounding boxes and the number of anchors. All five versions of the YOLOv5 models comprise the same backbone, neck, and head. The activation function was modified further, and two activation functions were added. The SiLU, also known as Swish, a Sigmoid Linear Unit, was added to hidden layers. The other activation function added at the output layer was the sigmoid function. YOLOv4 and YOLOv5 do not use fixed layers for object detection, but rather the network structure determines the detection layers [5].

In all YOLO models, no max pooling is applied. Instead, it performs convolution with a stride 2 for downsampling as the image loses low-level features in max pooling and does not lead to detecting smaller objects. Max pooling takes the most significant values from the 4 pixels, as shown in Figure 2, while convolutional pooling considers each pixel when performing the convolution, as shown in Figure 3.

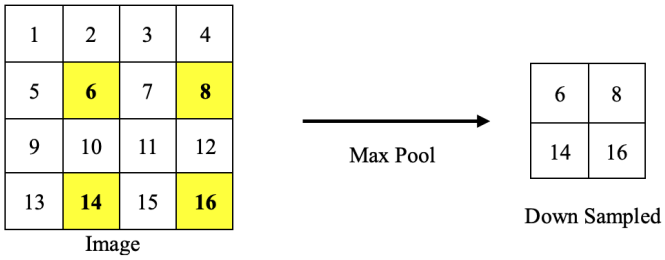


Fig. 2. Max Pooling Example

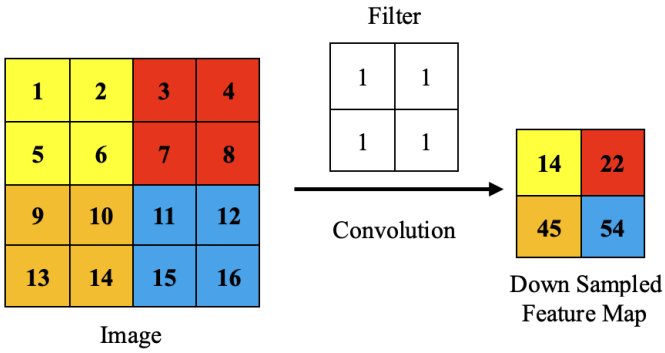


Fig. 3. Convolution Example

The main task of the YOLO algorithm is to classify if the cell within the grid is at the object's center. During training, the model tries to create one bounding box for each object, and each cell will produce three bounding boxes while predicting whether the object is at the center of its bounding box.

A. Model Input

The image input to the model network differs from the initial image size, which is a predefined image and can be of any size the user prefers. The predefined user image is fed to the network to perform pre-processing tasks, including normalizing and resizing the image. It is good practice to provide the model with a suitable image size to allow for faster training for the model. The YOLOv5 model takes the image as the input and resizes it to the following format (n, 3, 640, 640), where each field stands for (Batch Size, Channels, width, height). The user can predefine the image size before training, but it must be divisible by 32 as the network downsamples the image by strides of 32, 16, and 8. However, increasing the image size can increase the model's accuracy as it may capture enhanced details and patterns in the image, while the trade-off is the increase in training time.

B. Object Detection

Object detection in YOLOv3 is implemented at three different layers: 82, 94, and 106. Each layer performs downsampling through convolution and a 1-by-1 kernel to generate three new feature maps. Assume an image of size 416 by 416 is inputted to the model; the feature maps generated after downsampling at layer 82 (scale 1) with a stride of 32 will generate a 13 by 13 feature map, at layer 94 (scale 2) with a stride of 16 will generate a 26 by 26 feature map, and lastly at layer 106 with a stride of 8 will create a 52x52 (scale 3) feature map.

Object detection for YOLOv4 and YOLOv5 is not detected at three fixed layers as in YOLOv3 but still implements three scales at various layers dependent on the network structure. For YOLOv5, the three feature maps are generated: P3, P4, and P5, and for an image size of 640 by 640, after downsampling, the feature maps will have a size of 80 by 80, 40 by 40, and 20 by 20, respectively. The models look to predict which class the object belongs to using bounding boxes at the cell level. The image is divided into an n-by-n grid, and each cell represents the probability of an object and class. The bounding boxes are created around an object where each cell in the grid is represented in the following format for all YOLO models, as shown in Figure 4.

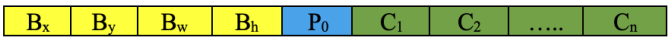


Fig. 4. Bounding Box Attributes

The following attributes represent the bounding box:

$$b_x, b_y, b_w, b_h$$

where

$$b_x, b_y$$

are the center coordinates of the predicted bounding box,

$$b_w, b_h$$

are the width and height of the predicted bounding box,

$$P_o$$

is the Objectness score (probability of an object within the bounding box), and

$$C_1, C_2, C_n$$

are the class probabilities or confidences for n classes [3].

Assuming an image size of 640 by 640 and each model uses three bounding boxes at each cell, the total bounding boxes at each scale are shown in Table 1. The total predicted bounding boxes are 25,200, where many of the bounding boxes are filtered out by applying Non-Maximum Suppression, which leaves only the bounding boxes containing an object.

TABLE I
BOUNDING BOXES AT EACH SCALE

Scale	Width	Height	Total Bounding Boxes
Scale 1	20	20	1200
Scale 2	40	40	4800
Scale 3	80	80	19200
Total			**25200**

The following equation calculates the shape or depth of the detection kernel.

$$d = b * (4 + 1 + c)$$

Where d is the depth of the detection kernel, b is the number of bounding boxes, 4 is the total number of bounding box attributes, 1 is the objectness score, and c is the number of classes.

The dimensions for each cell for the emotion recognition model are as follows: 7 emotion classes and three bounding boxes at each cell. The total depth of the detection kernel is,

$$d = 3 * (5 + 7) = 36$$

The output of the model at each detection layer will have a total attribute of 36 or tensor of shape (n, n, 36), (n, n, 36), and (n, n, 36), where n is the size of the feature map.

For YOLOv3 and YOLOv4, the bounding box coordinates are predicted using the following equations,

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w * e^{t_w}$$
$$b_h = p_h * e^{t_h}$$

Where

$$b_x, b_y$$

are the center coordinates of the predicted bounding box,

$$b_w, b_h$$

are the width and height of the predicted bounding box,

$$c_x, c_y$$

are the coordinates of the cell which are normalized to the actual image size.

$$p_w, p_h$$

are the anchor box dimensions normalized to the actual image size [3].

There is a slight modification in YOLOv5 for computing bounding box coordinates, as shown below [4],

$$b_x = 2 * \sigma(t_x) - 0.5 + c_x$$
$$b_y = 2 * \sigma(t_y) - 0.5 + c_y$$
$$b_w = p_w * 2 * \sigma(t_w)^2$$
$$b_h = p_h * 2 * \sigma(t_h)^2$$

C. Anchor Boxes

Anchors are predefined bounding boxes with various sizes that detect the probability of a cell containing an object. The anchors are computed before the model begins training using K-Means clustering on a dataset to get the predefined anchor boxes. They are used to calculate the predicted bounding box values for width and height. Three anchor boxes are used for YOLOv3 for each scale at layers 82, 94, and 106 for a total of 9 anchors. Since each scale implements three anchors, it will produce three bounding boxes at each cell while computing the maximum probability of the class confidence among the three bounding boxes.

In YOLOv3, it is predefined using the COCO dataset, while in YOLOv4 and YOLOv5, it uses the custom training dataset to get the predefined anchor boxes. YOLOv5 is more versatile in defining the total anchors initialized and allows for a more automated process than YOLOv4.

They are also used to calculate the predicted bounding box's real values for width and height while calculating the offset to the anchors, also known as Log-Space Transform. The predicted centre coordinates for each bounding box output are sent through a sigmoid function to get the probability between 0 and 1.

The network produces one ground truth box and one cell for each object while the model weights are trained using Mean Square Error as the loss function to predict bounding box coordinates for YOLOv3. YOLOv4 and YOLOv5 use a new loss function called GIoU (Generalized Intersection over Union) to calculate the loss, which is set as the default loss function for YOLOv5. GIoU is a modified version of IoU (Intersection over Union) to include more details for the bounding boxes, such as the overlap, distance, and aspect ratio. It performs better for ground truth bounding boxes and predicted bounding boxes that do not overlap, leading to better localization and detection. The GIoU loss function is calculated by the following equation [6],

$$LossFunction = 1 - GIou$$

Where

$$GIoU = IoU + \frac{AreaofBoundingBox - AreaofUnion}{AreaofBoundingBox}$$

and

$$IoU = \frac{Areaof Overlap}{Areaof Union}$$

After training, the network outputs the coordinates to calculate the width and height. The model does not calculate absolute values but uses offset from the centroid of the anchors, as it can help eliminate unstable gradient issues, such as vanishing gradients, during training. To get the actual values after prediction, it multiplies the width and height of the whole image to produce the coordinates for the bounding boxes.

D. Loss Function

In the table below, all the loss functions for YOLOv3, YOLOv4, and YOLOv5 are summarized, where MSE is Mean Square Error, BCE is Binary Cross Entropy, and GIoU is Generalized Intersection over Union.

TABLE II
LOSS FUNCTIONS

Loss	YOLOv3	YOLOv4	YOLOv5
Classifcation Loss	MSE	BCE	BCE
Bounding Box Loss	MSE	GIoU	GIoU
Objectness Loss	MSE	BCE	BCE

E. Objectness Score

Each bounding box is associated with an Objectness score, P0, which determines the bounding box probability of how close the cell is to the object's center and whether it contains an object. If P0 is close to 1, it is determined that the cell is at the center and contains an object, and if P0 is close to 0, it is the furthest away from the object's center, such as corners, and no object will be detected. The confidence score differs from the Objectness score as it determines the probability of which class the object belongs to rather than the probability of being at the object's center. Objectness score is calculated by multiplying the probability of an object within the bounding box and the intersection over union (IoU).The following equation calculates the Objectness score,

$$P_{Object} * IoU$$

$$P_{Object} = \sigma(t_o)$$

The output is passed through a sigmoid function to get the probability between 0 and 1 to determine if there is an object within the bounding box.

VI. Methodology and Implementation

A. Pre-Proccessing

The current dataset used for training and testing was downloaded from Roboflow, which contained a total of 2295 images. The images and labels were downloaded and uploaded to Google Drive to initialize the dataset. The configure data.YAML file included the paths to the training, validation, and testing images. It also included the total number of classes and class names included in the dataset provided for training.

The format of the labels in YOLOv5 format is defined as follows,

$$[class\_id, center\_x, center\_y, width, height]$$

where class_id is the class the image belongs to, center_x and center_y are the center coordinates of the bounding box, and width and height correspond to the bounding box. Each image in the dataset had a corresponding label in the label folder with the same name as the image but in .txt format. Before data augmentation was applied, the dataset was resized to 640 by 640 to match the input size of the model, and auto orientation was applied to the images, which allowed the images to be rotated to the correct orientation.

Data augmentation was only applied to the training dataset, not the validation or testing dataset, using Roboflow, to increase the size of the dataset to produce a generalized model and prevent overfitting. After data augmentation, the dataset contained 6000 images for training, 199 for validation, and 99 for testing. Each data augmentation technique was applied at a random rate with specified minimum and maximum values. The data augmentation techniques applied to the dataset are shown in Table 3.

TABLE III
DATA AUGMENTATION TECHNIQUES

Data Augmentation	Minimum	Maximum
Crop	0%	100%
Flip	0%	100%
Rotate	-8°	+8°
Shear	-8°	+8°
Blur	0%	100%
Noise	0%	100%

B. Model

The model implemented in training and testing was YOLOv5s (small), the most miniature model in the YOLOv5 family and optimized for speed and accuracy. The model consists of a backbone, neck, and head. The backbone is implemented to extract features from the input image using CSP-Darknet53 and consists of 10 layers with 5 CBS (Convolutional Block Stack includes a convolutional layer, batch normalization, and a SiLU activation function), 4 C3 layers, and 1 SPPF (Spatial Pyramid Pooling Fast) layer. The stacking of the CBS layers is used to help C3 layers learn complex features while reducing the number of parameters. The SPPF layer extracts features at different scales and sizes and fuses them. The last two C3s from the backbone are combined with the next C3 layer and the output of the SPPF layer in the neck to generate the feature maps. The neck generates three feature maps: P3, P4, and P5, with sizes 80 by 80, 40 by 40, and 20 by 20, respectively. The backbone layers are summarized in Table 4.

TABLE IV
BACKBONE LAYERS

Layer	Type	Filters	Size
0	Convolutional	32	6 by 6, stride 2
1	Convolutional	64	3 by 3, stride 2
2	C3	64	3 x (Conv + BatchNorm + SiLU)
3	Convolutional	128	3 by 3, stride 2
4	C3	128	3 x (Conv + BatchNorm + SiLU)
5	Convolutional	256	3 by 3, stride = 2
6	C3	256	3 x (Conv + BatchNorm + SiLU)
7	Convolutional	512	3 by 3, stride = 2
8	C3	512	3 x (Conv + BatchNorm + SiLU)
9	SPPF	512	2 x (Conv2d) + MaxPool2d

C. Training

The training dataset consisted of 6000 images with 7 classes and labels. The training took place on Google Colab using a GPU runtime and the YOLOv5s model with a batch size of 16 and 300 epochs. The total parameters for the model were 7,038,508, with 214 layers. The optimizer used for training was the default YOLOv5s optimizer, SGD (Stochastic Gradient Descent), with a learning rate of 0.01 and weight decay = 0.0005. The model picked up a few duplicates of the images and removed them from the dataset. The training took approximately 3.361 hours using a GPU runtime on Google Colab, and early stopping was applied to prevent overfitting, as there was no improvement in the last 100 epochs. The best parameters and weights were saved as best.pt. The training results and the figures for each metric are shown in the Results section, and the results are discussed further in the Discussion section.

VII. RESULTS

The training results are shown in Table 5, and the figures for each metric are shown in Figures 5 and 6. Figures 7 and 8 display the precision-confidence and recall-confidence curves, respectively. They are used to view the model's performance at different confidence thresholds to determine the optimal threshold for the model. The training results show that the model achieved a high accuracy with a precision of 86.4%, recall of 90.4% and mAP of 87.6% on the training dataset. This may have been due to the data augmentation techniques that allowed the model to learn additional image features and patterns, improving accuracy.

TABLE V
TRAINING RESULTS

Class	Precision	Recall	mAP 0.5	mAP 0.5:0.95
Anger	0.809	0.957	0.977	0.877
Contempt	1.000	0.753	0.981	0.957
Disgust	0.836	1.000	0.986	0.891
Fear	1.000	0.668	0.979	0.827
Happy	0.940	1.000	0.995	0.926
Sadness	0.910	0.950	0.965	0.865
Surprise	0.555	1.000	0.940	0.789
All	0.864	0.904	0.975	0.876

Fig. 5. Precision Curve

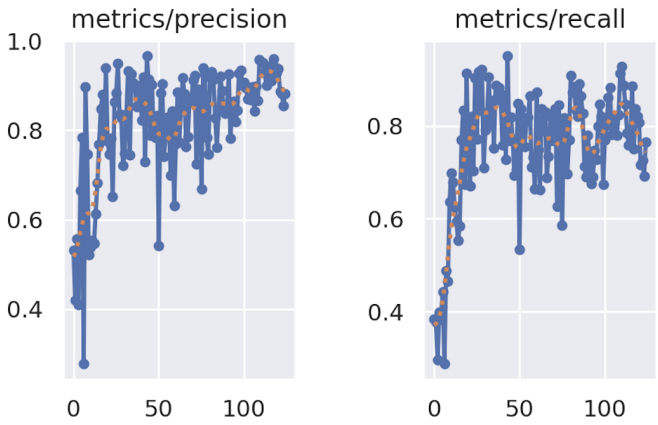


Fig. 6. Recall Curve

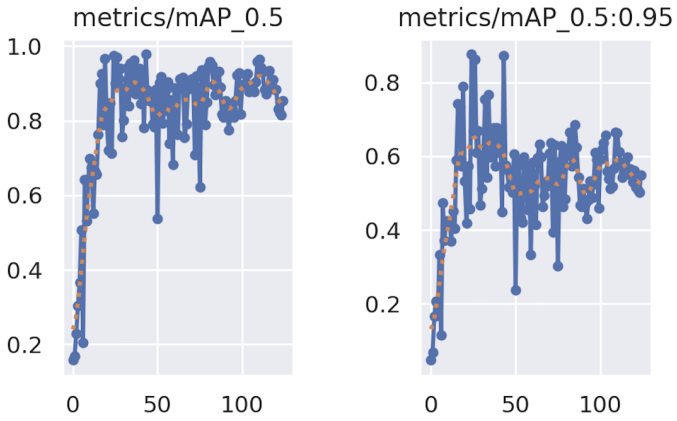


Fig. 7. Precision-Confidence Curve

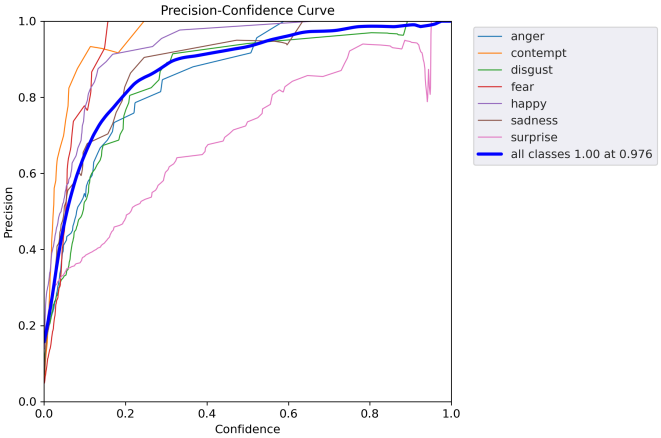
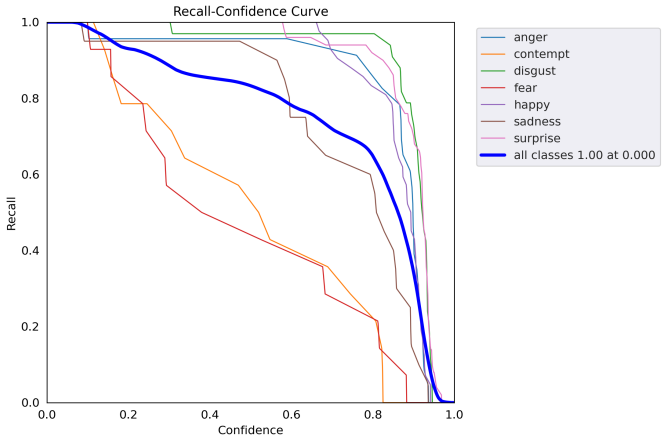


Fig. 8. Recall-Confidence Curve



A confusion matrix was created to visualize the model's performance on the training dataset, as shown in Figure 9. The confusion matrix shows the number of correct predictions in a diagonal line from the top left to the bottom right. The fear class had the lowest detection rate at 36%, followed by contempt at 71%. The model had difficulty differentiating between fear and surprise but could detect all the surprise classes. As for contempt, it also had trouble distinguishing between contempt and surprise, but it could detect the contempt class at a higher rate than the fear class.

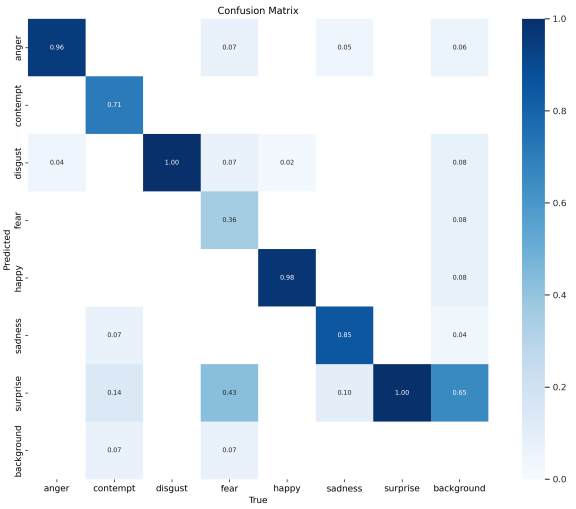


Fig. 9. Training Confusion Matrix

A. Testing

The validation dataset contained 196 images and assessed the model's performance during training to prevent overfitting. The testing dataset was used to evaluate the model's accuracy and performance on unseen data with no data augmentation applied. It contained 99 images with seven classes and corresponding labels. The dataset was passed through the model to predict each image's bounding boxes and class labels while computing various metrics to evaluate the model's performance. To comprehend how yolov5s detect objects in an image, each layer in the model needs to be analyzed from the backbone, neck, and head. This will allow us to understand better the model's architecture and why it is preferred for object detection. One test image was sent through the model, and the first five feature maps were extracted at each layer. Table 6 shows the Precision, Recall, and mAP metrics for the Testing dataset, and a confusion matrix is provided in Figure 10. Furthermore, the total accuracy of the model was computed for the testing dataset, and it was approximately 92.9%

TABLE VI
TESTING RESULTS

Class	Instances	Precision	Recall	mAP 0.5
Anger	14	0.909	1.000	0.990
Contempt	6	1.000	0.767	0.917
Disgust	22	0.974	1.000	0.995
Fear	5	1.000	0.745	0.900
Happy	25	1.000	1.000	0.995
Sadness	7	0.981	0.857	0.918
Surprise	20	0.837	0.950	0.919
All	99	0.957	0.903	0.948

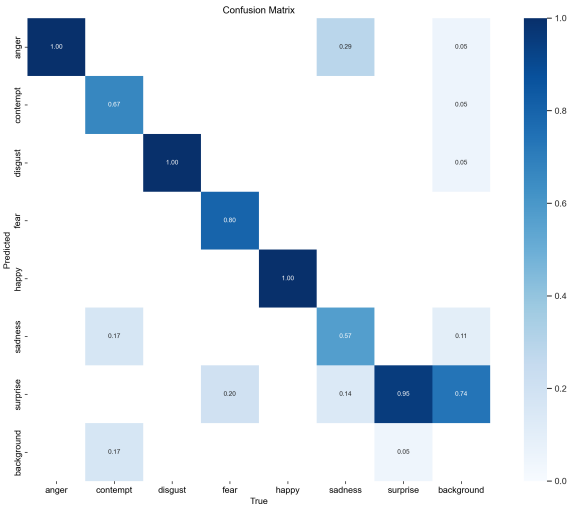


Fig. 10. Testing Confusion Matrix

Calculating the total accuracy through the confusion matrix is shown below,

$$Total Accuracy = 1.00 * (14) + 0.67 * (6) + 1.00 * (22) +$$
$$0.8 * (5) + 1.00 * (25) + 0.57 * (7) + 0.95 * (20)/99 = 0.929$$

B. Backbone Module

The backbone layers extract features from the input image and generate the feature maps for the Neck module. The backbone layers, CSP-Darknet53, consist of 10 layers with 5 CBS (Convolutional Block Stack includes a Convolutional layer, Batch normalization, and a SiLU activation function), 4 C3 layers, and 1 SPPF (Spatial Pyramid Pooling Fast) layer.

The first five filters of each layer are displayed to analyze and understand how the features are extracted from the input image. The first two layers are shown in Figures 11 and 12.

Fig. 11. Backbone Layer 0

Fig. 12. Backbone Layer 1

The first two layers are the convolutional layers with a kernel size of 3 and use a stride of 2. The first layer has 64 filters, and the second layer has 128 filters. Each Convolution Block goes through a batch normalization and SiLU activation function. The output of the second layer goes through a C3 layer to extract features from the image.

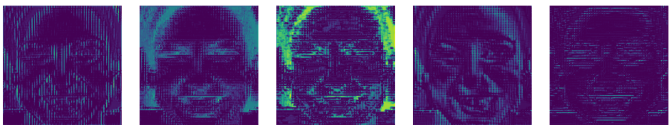


Fig. 13. Backbone Layer 2

Two C3 layers and 1 SPPF layer are passed to the neck to extract features from the image. These layers are 4, 6, and 9, as shown in Figures 15, 17, and 20.

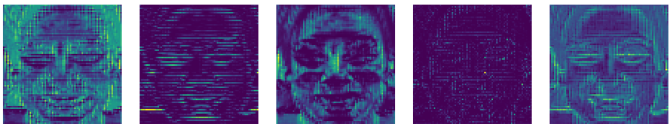


Fig. 14. Backbone Layer 3

Fig. 15. Backbone Layer 4

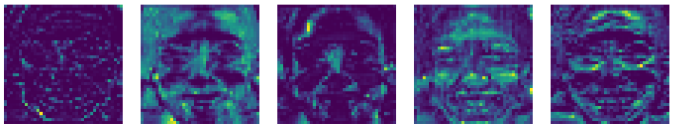


Fig. 16. Backbone Layer 5

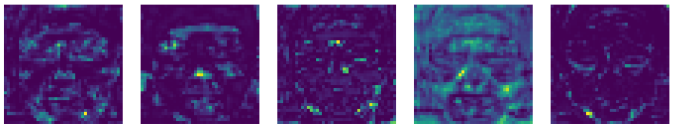


Fig. 17. Backbone Layer 6

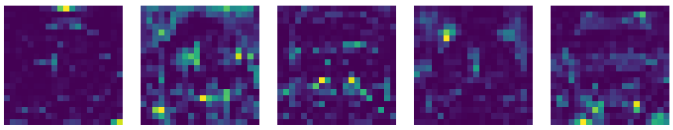


Fig. 18. Backbone Layer 7

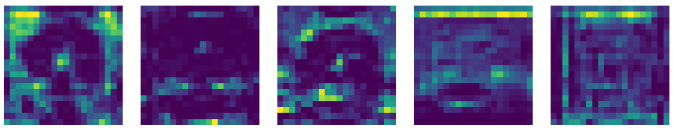


Fig. 19. Backbone Layer 8

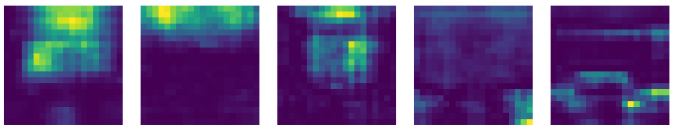


Fig. 20. Backbone Layer 9

C. Neck Module

The Neck Module are used to refine the features extracted from the backbone layers to generate feature maps for the Head Module to allow object detection of various sizes (small, medium, or large). It consist of a PAN and contain a total of fourteen layers as shown below. Images are concatenated together to fuse features from different scale and sizes to allow the model to learn complex patterns and features. The fourteen layers are shown in Table 7.

TABLE VII
NECK LAYERS

Layer	Type
10	Convolutional
11	Upsample
12	Concat
13	C3
14	Convolutional
15	Upsample
16	Concat
17	C3
18	Convolutional
19	Concat
20	C3
21	Convolutional
22	Concat
23	C3

D. Concatenation Layers

The concat layer takes the images and fuses features from different scales and sizes, allowing the model to learn complex patterns and features. The first concat layer takes layer six and concatenates it with layer 11. The second concat layer takes layer four and concatenates it with layer 15, the previous concat layer's output after convolution and upsampling. The third concat layer takes layer 18 and concatenates it with layer 14, and lastly, the fourth concat layer takes layer 21 and concatenates it with layer 10. The feature maps of each concat layer after performing fusion are shown in Figures 21 to 24.

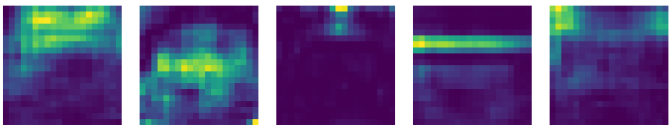


Fig. 21. Feature maps of first concatenation layer

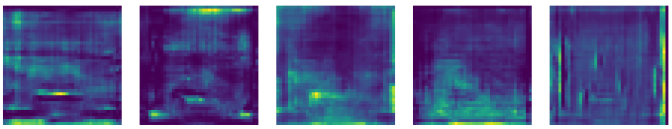


Fig. 22. Feature maps of second concatenation layer

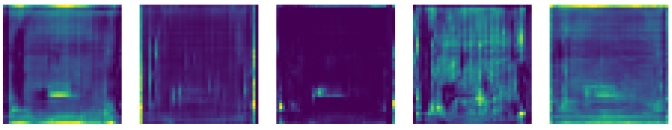


Fig. 23. Feature maps of third concatenation layer

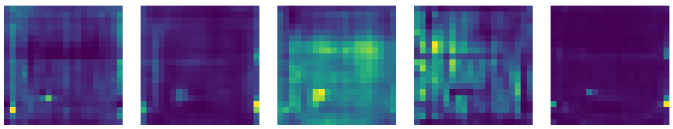


Fig. 24. Feature maps of fourth concatenation layer

E. Head Module

The head takes three feature maps generated from layers 17, 20, and 23 with various channels from the neck and performs Convolution with kernel size and stride of 1 to produce 128, 256, and 512 feature maps with 36 attributes for each feature map. These three feature maps of various channels are used to detect small, medium, and large objects. Lastly, the three generated feature maps and the output of the head are shown in Figures 25 to 27, and the final output of the model is shown in Figure 28.

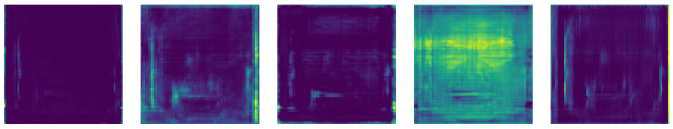


Fig. 25. Head Feature Map 1

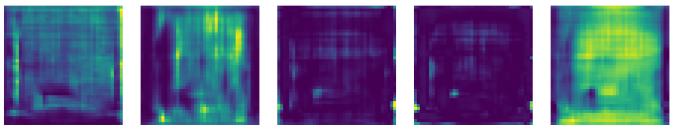


Fig. 26. Head Feature Map 2

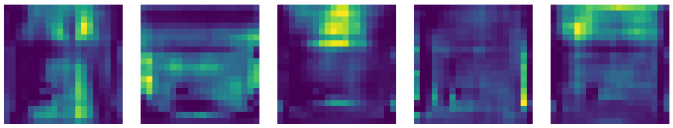


Fig. 27. Head Feature Map 3

Fig. 28. Head Layer Output with Bounding Boxes and Class Labels

VIII. Discussion

The model produced satisfactory results on the training dataset with precision of 86.4%, recall of 90.4% and mAP of 87.6% on the training dataset. The model detected the emotions in the images with high accuracy even though the model was trained on a small dataset. The model may have produced better results due to adding data augmentation to the training dataset, which allowed the model to be more generalized and prevent overfitting. The model was also trained on a GPU runtime on Google Colab Pro, which allowed the model to train faster and more efficiently. The YOLOv5 model outperformed the I-ML model discussed in class as it could detect emotions in images with higher accuracy, precision, and recall. It confirms deep learning is one of the best approaches for feature extraction for most image classification tasks and can be used for emotion detection, as confirmed by the results.

A few issues were encountered during the project, mainly trying to implement a real-time emotion detection system. The current laptop I have been using is a MacBook Pro with a new architecture, ARM, which does not support the CUDA architecture required to run the model on a GPU. I ran into many issues trying to install the CUDA toolkit and PyTorch on my laptop, which led to the kernel crashing and restarting. Therefore, I was unable to implement a real-time application. Another unexpected issue was the background information in the images fed to the model. All the images in the training dataset were taken in a controlled environment with no variation in the background, and the images were aligned perfectly with the face. However, images were taken from the webcam and sent through the model to determine whether the model could detect the emotions in the images. The results varied as the images contained various patterns and features in the background, which the model needed to be trained on, leading to incorrect predictions and lower accuracy. To counter this issue, the images sent to the model must be cropped to include only the face with all the background noise removed, or a new dataset with complex backgrounds must be used to train the model. In Figures 29 and 30, the model picked up a a couple of emotions and even though it classified the emotion correctly, it created multiple bounding boxes around the face, which may have been due to the background noise in the images as there was a divider in the background. In Figures 31 the model was unable to detect the emotions in the images at all and misclassified the emotions but using a different image with no background noise, the model was able to detect the emotions correctly as shwn in Figure 32.

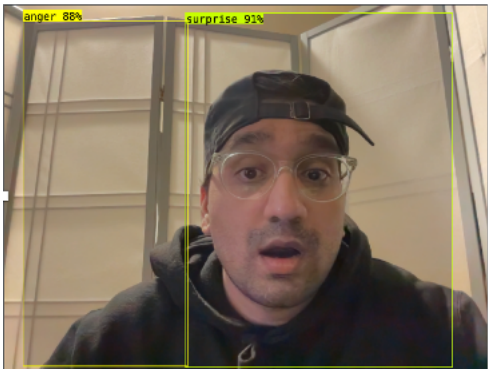


Fig. 29. Real Time Emotion Detection from Webcam Example 1

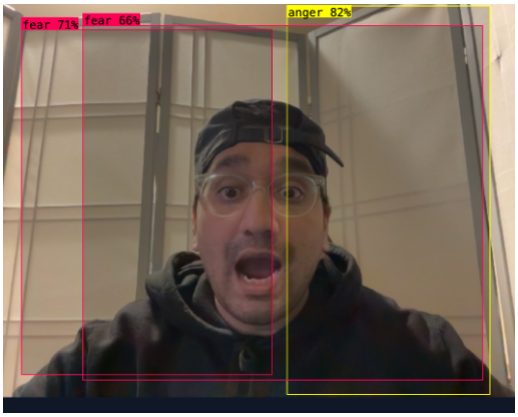


Fig. 30. Real Time Emotion Detection from Webcam Example 2

Fig. 31. Static Image Example 1

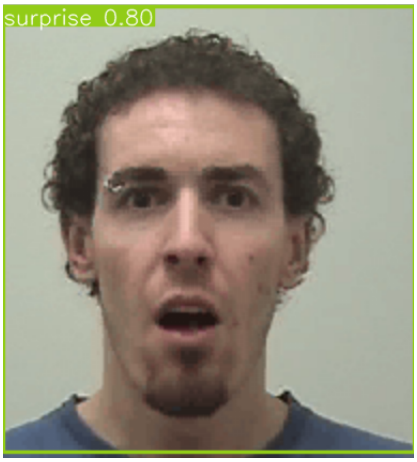


Fig. 32. Static Image Example 2

IX. Conclusion

After a better understanding of the model's architecture, a few improvements can be made to improve the model's accuracy. Although I still believe Deep Learning is the best approach for feature extraction for emotion recognition, further research must be done to determine the best architecture for emotion recognition. In future work, I would like to implement the application on a GPU-acceptable device to test the model in real-time, as I was currently unable to set up an environment locally or through Google Colab due to webcam setup in the cloud and the lack of compute units available for the GPU runtime. Nonetheless, the model produced acceptable results but can be optimized further to improve the accuracy.

References

[1] [1] The Centre for Addiction and Mental Health — Camh, https://www.camh.ca/-/media/files/pdf—osduhs/2021-osduhs-report-pdf.pdf (accessed Oct. 27, 2023).

[2] [2] A. Atam, "Architecture of yolov3," OpenGenus IQ: Computing Expertise & Legacy, https://iq.opengenus.org/architecture-of-yolov3/ (accessed Dec. 9, 2023).

[3] [3] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," arXiv.org, https://arxiv.org/abs/1804.02767 (accessed Dec. 8, 2023).

[4] [4] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal Speed and accuracy of object detection," arXiv.org, https://arxiv.org/abs/2004.10934v1 (accessed Dec. 10, 2023).

[5] [5] C. Imane, "Yolo V5 model architecture [explained]," OpenGenus IQ: Computing Expertise & Legacy, https://iq.opengenus.org/yolov5/ (accessed Dec. 12, 2023).

[6] [6] H. Rezatofighi et al., "Generalized intersection over union: A metric and a loss for bounding box regression," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2019. doi:10.1109/cvpr.2019.00075 (accessed Dec. 12, 2023).