Daniel Elbachá Marnet delmarnet1@gmail.com

1. Introdução

O objetivo do presente trabalho foi conduzir um pequeno experimento para avaliar a melhor combinação entre hardware e software em busca de acurácia alta e de tempo de inferência baixo para aplicações em sistemas embarcados para classificação de rótulos de cerveja. Os métodos tradicionais de detecção de objetos usam algoritmos de inteligência artificial que necessitam de hardwares com elevada capacidade de processamento, os quais não são adequados para aplicação em sistemas embarcados voltados para detecção de objetos em tempo real, em campo, devido a restrições de tamanho, peso e energia [1].

Mazzia et al. (2020) apresentaram um estudo com uma solução embarcada em tempo real inspirada em ''Edge AI'' proposta para detecção de maçãs com a implementação do algoritmo YOLOv3-tiny, em várias plataformas embarcadas, como Raspberry Pi 3 B+ em combinação com Intel Movidius Neural Computing Stick (NCS), Jetson Nano da Nvidia e Jetson AGX Xavier [1]. Com base nessa proposta, foram analisadas bibliografias voltadas para a detecção de objetos em tempo real, e assim foi verificado o artigo original sobre o modelo YOLO [2] e um artigo de revisão abordando sobre diversas modificações feitas na rede YOLO para melhoria da eficiência da detecção de objetos [3]. Também, conforme visto que [1] discutiu sobre outros modelos voltados para a detecção e segmentação de imagens em tempo real, tal como Faster R-CNN, foi assim também analisada a arquitetura do modelo Mask R-CNN, do mesmo autor da Faster R-CNN, para avaliar as diferenças desses modelos e as formas para a adaptação dos usos voltados para detecção e segmentação de imagens [4].

Com base nessas buscas bibliográficas, foi constatado no artigo de revisão [5] que a classificação de objetos em tempo real pode ser obtida com boa eficiência e acurácia aplicando modelos *Convolutional Neural Network* (CNN). Assim, o presente trabalho utilizou a biblioteca *open source* Keras TensorFlow para gerar dois modelos CNN, um modelo com uma arquitetura mais simplificada, e um segundo modelo mais complexo em questão de quantidade de camadas e de parâmetros, para avaliar e comparar as diferenças entre os desempenhos desses.

2. Metodologia

Inicialmente foram realizados os downloads e as listagens dos arquivos das imagens da Ambevtech, e após isso o dataset inicial foi dividido em três conjuntos: treino (com 60% dos arquivos), validação (20%) e teste (20%). Todos os arquivos foram processados para normalização dos valores dos pixels e também para padronização das dimensões em 128x128.

Após o tratamento e divisão dos dados das imagens, foram geradas as camadas dos modelos de redes neurais artificiais. Ambos os modelos elaborados são objetos sequenciais do Keras TensorFlow voltados para resolução de classificação (5 classes de rótulos de cerveja). Para a classificação final das probabilidades dos resultados obtidos, foi utilizada em ambos os modelos a função de ativação softmax. O primeiro modelo foi gerado com as camadas e o total de parâmetros conforme indicado na figura resumo abaixo: Modelo 1.

Modelo 1

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 128)     3584

 max_pooling2d (MaxPooling2D  (None, 63, 63, 128)      0
 )

 dropout (Dropout)           (None, 63, 63, 128)       0

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        73792

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 30, 30, 64)        0

 conv2d_2 (Conv2D)           (None, 28, 28, 32)        18464

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 dropout_2 (Dropout)         (None, 14, 14, 32)        0

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 577)               3619521

 dropout_3 (Dropout)         (None, 577)               0

 dense_1 (Dense)             (None, 577)               333506

 dropout_4 (Dropout)         (None, 577)               0

 dense_2 (Dense)             (None, 5)                 2890

=================================================================
Total params: 4,051,757
Trainable params: 4,051,757
Non-trainable params: 0
```

O modelo 1 foi processado com 100 épocas e computou o tempo para treinar de 560.8 segundos. O modelo 2 foi processado com 300 épocas e computou o tempo para treinar de 1756.5 segundos. O modelo 2, além de possuir uma arquitetura mais complexa e de exigir maior custo de memória, necessitou o triplo do número de épocas do modelo 1 para conseguir resultados similares.

O modelo 2 foi baseado numa arquitetura voltada para a detecção de emoções em faces de pessoas [6]. Essa arquitetura foi selecionada por cumprir uma tarefa de classificação de imagens de complexidade elevada. Esse segundo modelo foi gerado com as camadas e o total de parâmetros conforme indicado na figura resumo abaixo: Modelo 2.

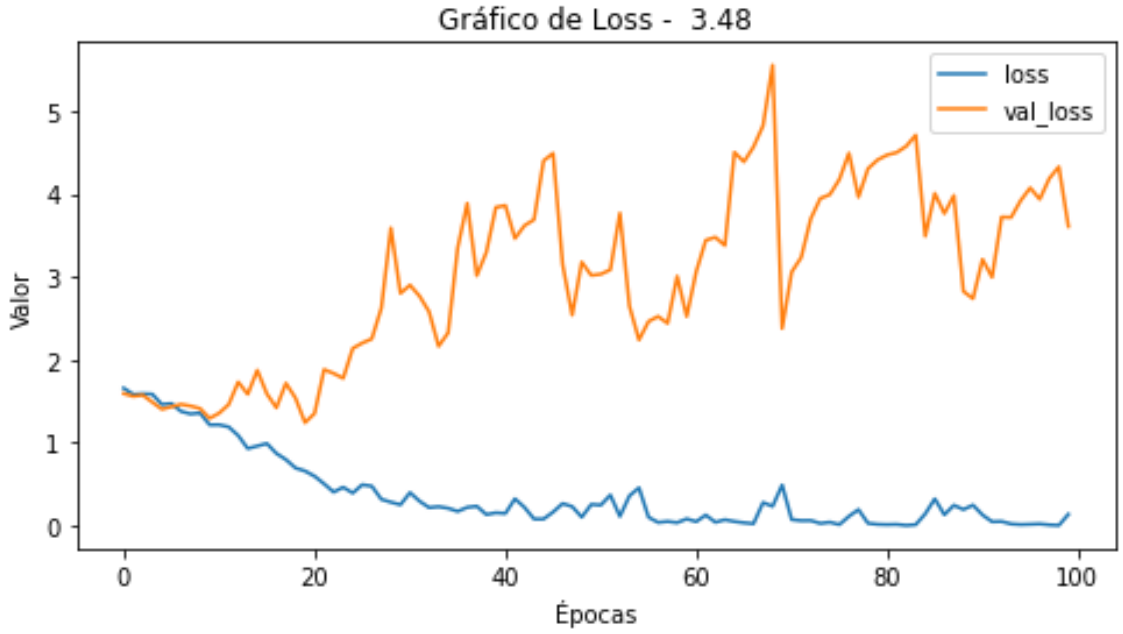
Modelo 2

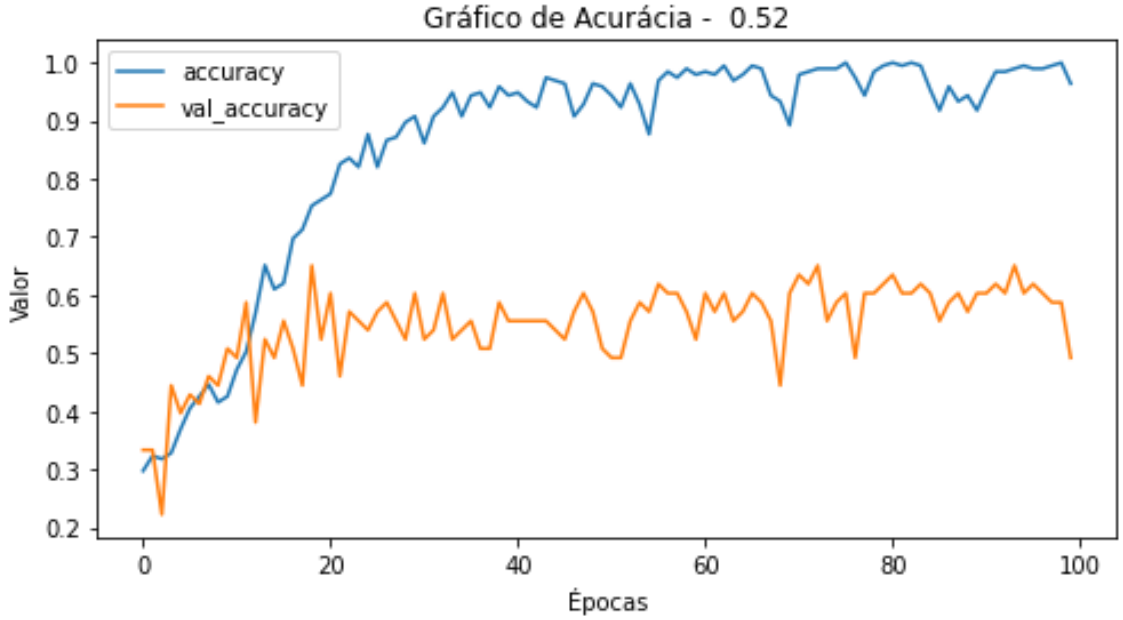
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 128, 128, 32)      896

 batch_normalization (BatchN  (None, 128, 128, 32)     128
 ormalization)

 conv2d_4 (Conv2D)           (None, 128, 128, 32)      9248

 batch_normalization_1 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 dropout_5 (Dropout)         (None, 64, 64, 32)        0

 conv2d_5 (Conv2D)           (None, 64, 64, 64)        18496

 batch_normalization_2 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 conv2d_6 (Conv2D)           (None, 64, 64, 64)        36928

 batch_normalization_3 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 dropout_6 (Dropout)         (None, 32, 32, 64)        0

 conv2d_7 (Conv2D)           (None, 32, 32, 128)       73856

 batch_normalization_4 (Batc  (None, 32, 32, 128)      512
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 dropout_7 (Dropout)         (None, 16, 16, 128)       0

 conv2d_9 (Conv2D)           (None, 16, 16, 256)       295168

 batch_normalization_6 (Batc  (None, 16, 16, 256)      1024
 hNormalization)

 conv2d_10 (Conv2D)          (None, 16, 16, 256)       590080

 batch_normalization_7 (Batc  (None, 16, 16, 256)      1024
 hNormalization)

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 dropout_8 (Dropout)         (None, 8, 8, 256)         0

 flatten_1 (Flatten)         (None, 16384)             0

 dense_3 (Dense)             (None, 64)                1048640

 dense_4 (Dense)             (None, 64)                4160

 batch_normalization_9 (Batc  (None, 64)               256
 hNormalization)

 dropout_10 (Dropout)        (None, 64)                0

 dense_5 (Dense)             (None, 5)                 325

=================================================================
Total params: 2,229,477
Trainable params: 2,227,429
Non-trainable params: 2,048
```

3. Resultados

O modelo 1 obteve um valor de acurácia igual a 0.52. Nas figuras abaixo, mostra-se o histórico de treinamento e de validação do modelo 1

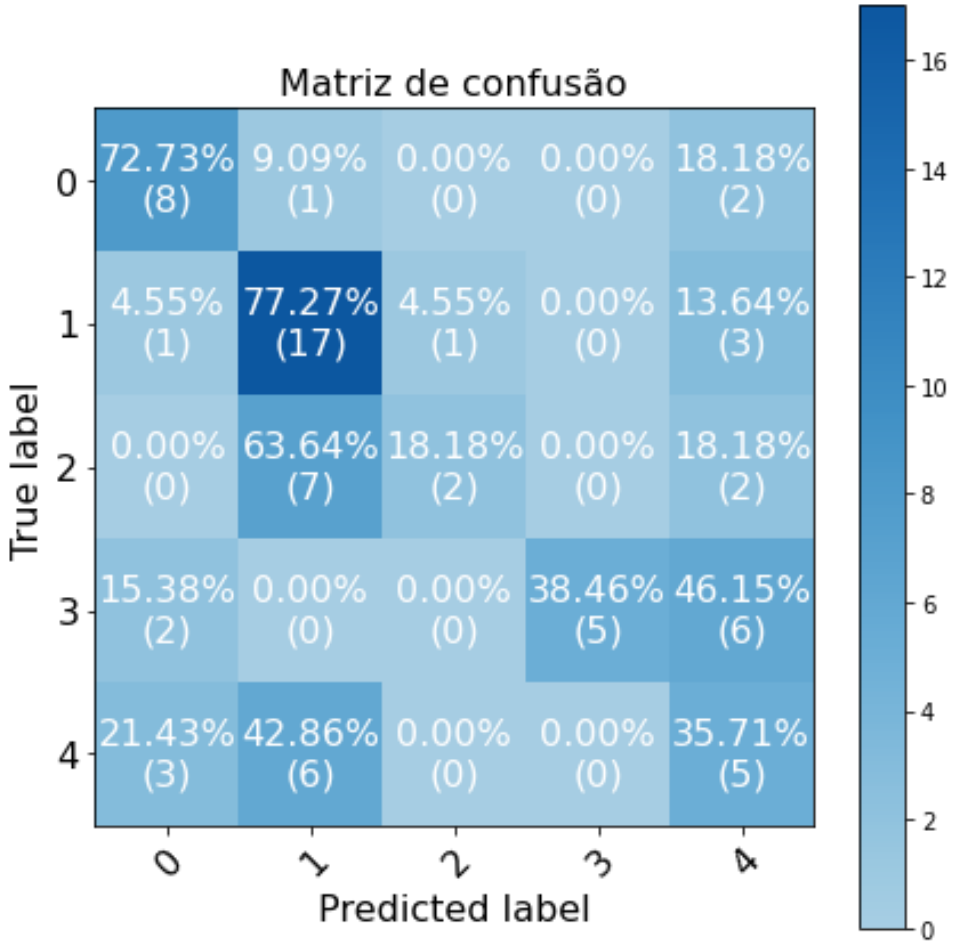
Histórico Modelo 1





Pode-se notar por esses gráficos, que foi gerado um treinamento com Overfitting, que ocorre quando os ajustes dos pesos são super ajustados aos dados de treinamento, diminuindo a capacidade de generalização do modelo, conforme constatado com os valores não satisfatórios para os dados de validação [7]. Para a otimização desse cenário foram adicionadas mais camadas de Dropout, visando aumentar a capacidade de generalização [7].

Matriz de Confusão do Modelo 1

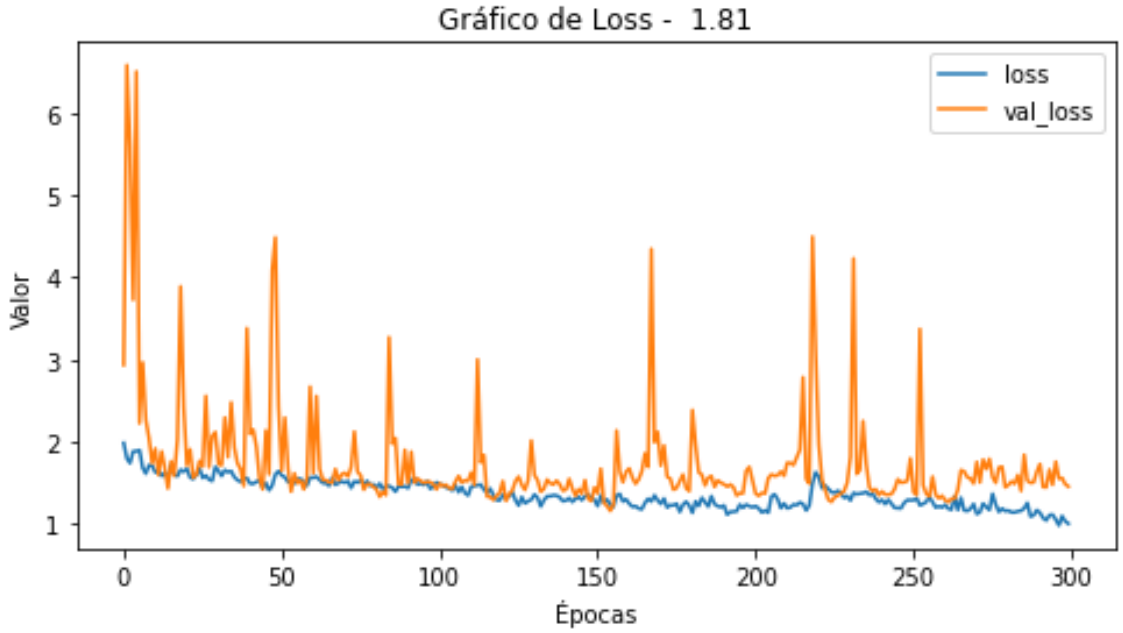


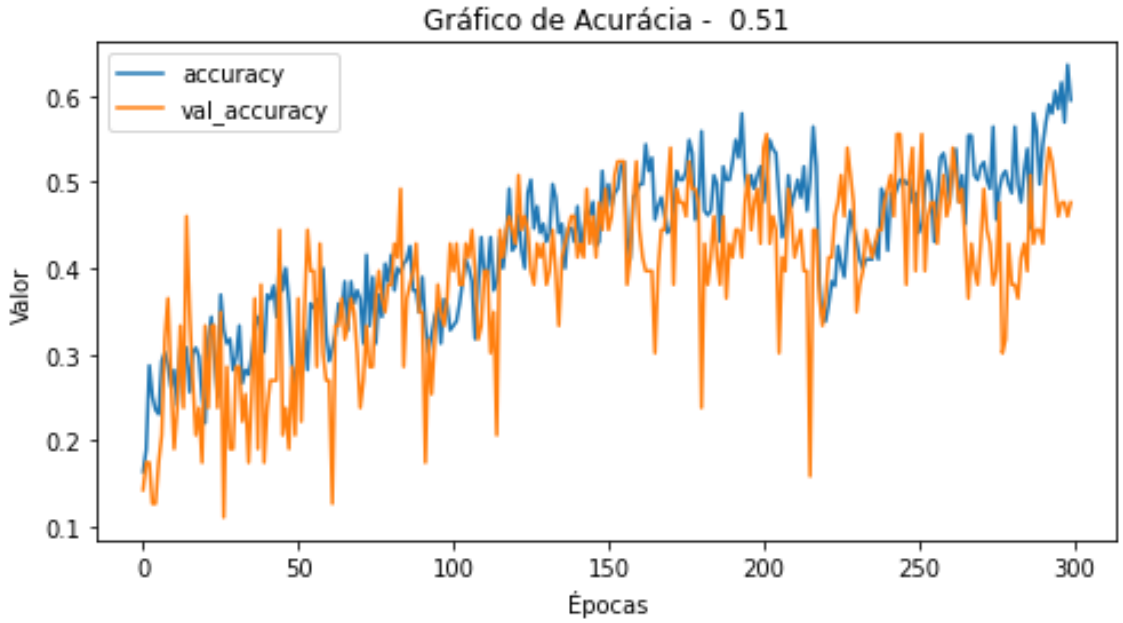
Resultados das Métricas de Classificação do Modelo 1

	Classes	F1	ROC AUC	PRC AUC	Precision	Recall	Specificity	Accuracy
0	cerveja Beck's	0.64	0.81	0.46	0.57	0.73	0.90	0.87
1	cerveja Brahma	0.64	0.74	0.49	0.55	0.77	0.71	0.73
2	cerveja BudWeiser	0.29	0.58	0.25	0.67	0.18	0.98	0.86
3	cerveja Corona	0.56	0.69	0.50	1.00	0.38	1.00	0.89
4	cerveja Skol	0.31	0.56	0.23	0.28	0.36	0.77	0.69
5	Média	0.49	0.68	0.39	0.61	0.48	0.87	0.81

O modelo 2 obteve um valor de acurácia igual a 0.51. Nas figuras abaixo, mostra-se o histórico de treinamento e de validação do modelo 2.

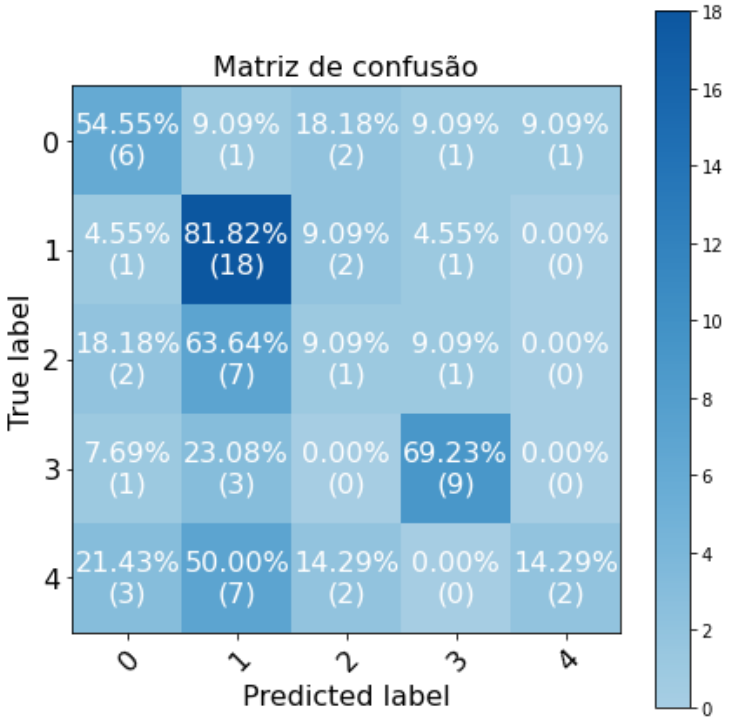
Histórico Modelo 2





O modelo 2 apesar de possuir maior custo computacional e de praticamente ter o mesmo valor de acurácia do modelo 1, nota-se nos gráficos dos históricos, melhor condição de generalização devido ao menor overfitting obtido no treinamento.

Matriz de Confusão do Modelo 2



Resultados das Métricas de Classificação do Modelo 2

	Classes	F1	ROC AUC	PRC AUC	Precision	Recall	Specificity	Accuracy
0	cerveja Beck's	0.50	0.71	0.32	0.46	0.55	0.88	0.83
1	cerveja Brahma	0.62	0.73	0.47	0.50	0.82	0.63	0.69
2	cerveja BudWeiser	0.11	0.50	0.15	0.14	0.09	0.90	0.77
3	cerveja Corona	0.72	0.82	0.58	0.75	0.69	0.95	0.90
4	cerveja Skol	0.24	0.56	0.26	0.67	0.14	0.98	0.82
5	Média	0.44	0.66	0.36	0.50	0.46	0.87	0.80

4. Conclusão

Conclui-se com essa pesquisa e com os resultados obtidos, código fonte com as implementações no github e README da implementação [8], que para sistemas embarcados voltados para a detecção de objetos em tempo real, pode-se aplicar modelos CNN, pois este modelo uma vez treinado com uma validação de alta acurácia e baixo overfitting, estará apto para classificar os objetos utilizando os pesos salvos. Ambos os modelos tiveram uma acurácia aproximadamente de 50% para classificação dos rótulos de cerveja em 5 classes. O modelo 1 apresentou a vantagem de necessitar um menor custo de memória e de tempo de processamento, e o modelo 2 apresentou a vantagem de possuir maior capacidade de generalização para classificar novos arquivos de imagens.

Referências

[1] V. Mazzia et al., “Real-Time Apple Detection System Using Embedded Systems With Hardware Accelerators: An Edge AI Application”, IEEE Access, Vol 8, Pag 9102 - 9114, 2020.

[2] J. Redmon et al., “You Only Look Once: Unified, Real-Time Object Detection", Cornell University, Vol 1, 2015.

[3] B. S. Rekha et al., “Literature Survey on Object Detection using YOLO ", International Research Journal of Engineering and Technology (IRJET), Vol 7, 2020.

[4] K. He et al., “Mask R-CNN", Cornell University, Vol 1, 2017.

[5] N. Aloysius; M. Geetha, “A review on deep convolutional neural networks", International Conference on Communication and Signal Processing (ICCSP), 2017.

[6] R. Ratan, “Using LittleVGG for Emotion Detection", <https://github.com/rajeevratan84/DeepLearningCV>.

[7] N. Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting ", Journal of Machine Learning, Vol 15, 2014.

[8] D. Marnet, “Daniel_Marnet_AmbevTech", <https://github.com/edanielmarnet/Daniel_Marnet_AmbevTech >.