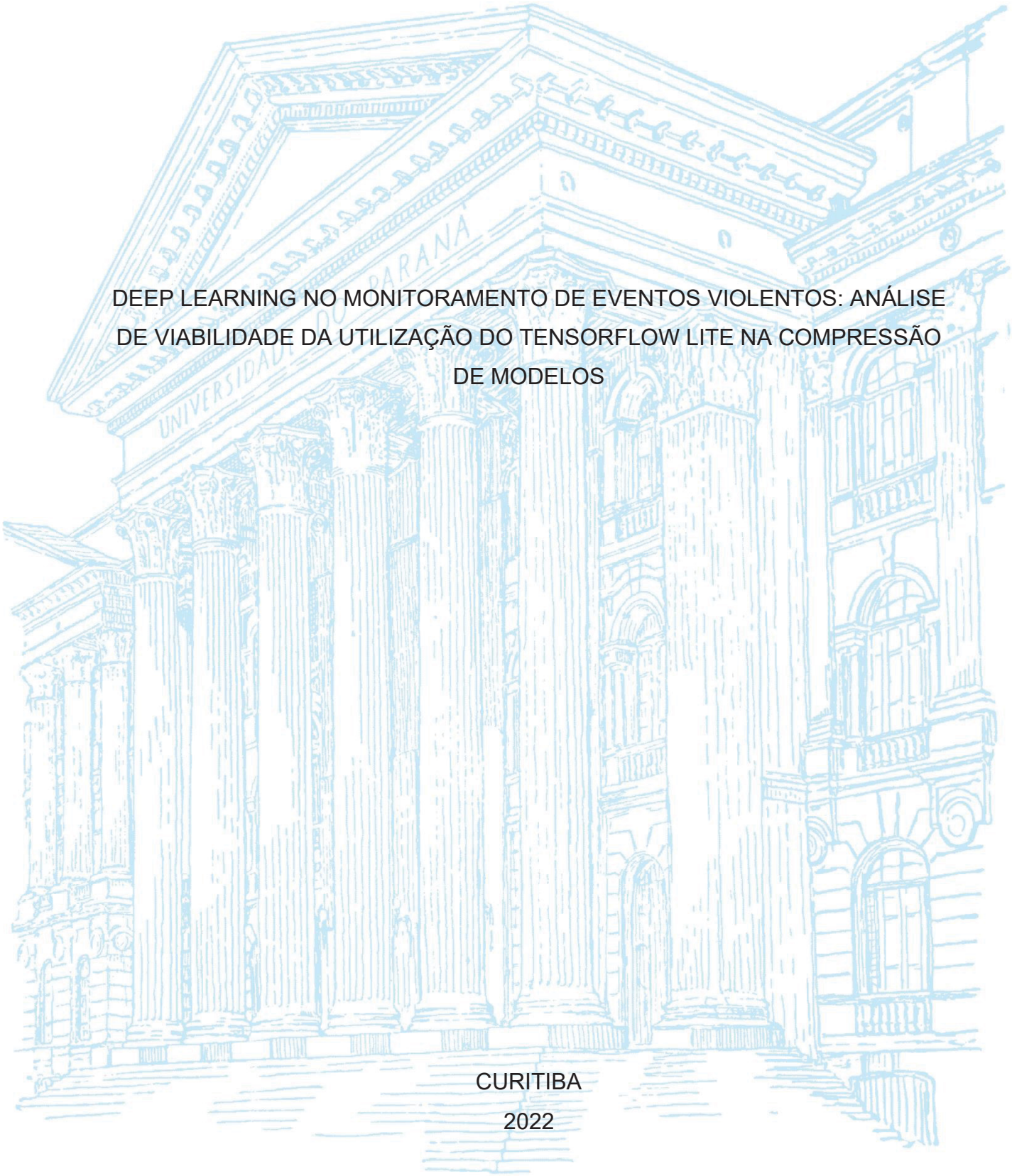
UNIVERSIDADE FEDERAL DO PARANÁ

GABRIEL GALPERIN



DEEP LEARNING NO MONITORAMENTO DE EVENTOS VIOLENTOS: ANÁLISE DE VIABILIDADE DA UTILIZAÇÃO DO TENSORFLOW LITE NA COMPRESSÃO DE MODELOS

CURITIBA

2022

GABRIEL GALPERIN

DEEP LEARNING NO MONITORAMENTO DE EVENTOS VIOLENTOS: ANÁLISE DE VIABILIDADE DA UTILIZAÇÃO DO TENSORFLOW LITE NA COMPRESSÃO DE MODELOS

Trabalho de conclusão de curso apresentada ao curso de Pós-Graduação em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Razer Anthom Nizer Rojas Montaño

CURITIBA

2022

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **GABRIEL GALPERIN** intitulada: **Deep Learning No Monitoramento De Eventos Violentos: Análise De Viabilidade Da Utilização Do Tensorflow Lite Na Compressão De Modelos**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 17 de Novembro de 2022.

RAZER ANTHOM NIZER ROJAS MONTAÑO
Presidente da Banca Examinadora

RAFAELA MANTOVANI FONTANA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Deep Learning no monitoramento de eventos violentos: Análise de viabilidade da utilização do *TensorFlow Lite* na compressão de modelos

Gabriel Galperin
Especialização em Inteligência Artificial Aplicada
Universidade Federal do Paraná (UFPR)
Curitiba, Brasil
galperin@ufpr.br

Razer Anthom Nizer Rojas Montaño
Especialização em Inteligência Artificial Aplicada
Universidade Federal do Paraná (UFPR)
Curitiba, Brasil
razer@ufpr.br

***Resumo*—Este trabalho apresenta um estudo de viabilidade na utilização do TensorFlow Lite na criação de modelos baseados em modelos existentes para a detecção de ações violentas em mídia de vídeo. A comparação entre os modelos foi feita utilizando a acurácia, precisão, revocação, tamanho do modelo e tempo de inferência sendo o modelo gerado a partir do MobileNetV2 e otimização de compressão Float16 o melhor encontrado. Este modelo apresenta revocação de 99.248% e compressão de mais de 50% comparado com modelo não comprimido.**

***Palavras-chave*—Violência, TensorFlow Lite, Compressão, Transfer Learning.**

***Abstract*—This paper present a viability study of using TensorFlow Lite to create models based in TensorFlow existing models to detect violence actions into video data. The comparison between models is done based on accuracy, precision, recall, model size and inference time, being the model created using MobileNetV2 and compression optimization Float16 the best found. This model presents recall of 99.248% and compression of more than 50% compared to uncompressed model.**

***Index Terms*—Violence, TensorFlow Lite, Compression, Transfer Learning.**

I. DESENVOLVIMENTO

O aumento no volume de atos de violência em espaços públicos resultou na ampliação da utilização de câmeras de monitoramento para auxiliar as autoridades na identificação de eventos e infratores [1]. Desta forma, com o intuito de manter a segurança pública, centenas ou milhares de equipamentos de monitoramento por vídeo são utilizados [2].

A forma mais comum de processamento das informações coletadas é com a utilização de inspeção humana, que se torna ineficiente e intangível devido ao alto volume de informação a ser analisado [1]. Outra forma de processar os dados coletados é através de algoritmos de processamento de imagem [1].

Estes algoritmos tem por objetivo detectar e classificar automaticamente ações de pessoas nos vídeos sob análise [2].

Apesar das vantagens que podem ser obtidas utilizando os algoritmos, ambas as abordagens de processamento apresentam uma deficiência compartilhada, a necessidade de transferir os dados de onde são obtidos para onde serão analisados.

Com o aumento da migração dos modelos de câmeras do analógicos para o digital na última década, a transmissão necessita da disponibilidade de *internet* [3].

Uma forma de diminuir a infraestrutura de apoio requisitada é realizar o processamento das informações diretamente no equipamento de coleta.

O processamento de dados diretamente nos dispositivos de coleta acarreta na necessidade de um equipamento mais sofisticado e robusto [3]. Estes equipamentos podem se tornar proibitivos tendo em vista o número de unidades necessárias para manter o nível de vigilância desejado.

Uma das formas de reduzir a necessidade de infraestrutura de apoio e o poder de processamento do equipamento de coleta é utilizando a compressão dos dados sendo processados e transmitidos.

Este trabalho apresenta um estudo de viabilidade onde o modelo utilizado na detecção de eventos violentos é comprimido utilizando a ferramenta *TensorFlow Lite*.

A. Descrição dos dados

Trabalhos visando identificar ações humanas cotidianas ganharam notoriedade nas comunidades de pesquisa científica, contudo pouco foi realizado na detecção de ações violentas [2].

Esta situação mudou com a criação de bases de dados focadas na diferenciação de atividades violentas e cotidianas [2].

Uma dessas bases de dados é a *Hockey Fight Videos* disponível em [4] e utilizada neste trabalho.

Nela 1000 vídeos do esporte *Hockey* são divididos em 500 amostras de eventos classificados como violentos e outros 500 como não violentos.

Todos os vídeos apresentam uma resolução de 360x288 *pixels* e a classificação dos dados entre as duas classes é realizada pela nomenclatura do arquivo.

A classe de vídeos contendo ações violentas são identificados por fi*_xvid, enquanto os que pertencem a classe não-violenta são identificados por no*_xvid, e o carácter * representa um contador único de cada vídeo variando de 1 a

500 com o intuito de permitir a diferenciação dos arquivos dentro da mesma classe.

Os vídeos demandam grande espaço de memória para serem processados, desta forma foram retiradas imagens que serão utilizadas de maneira representativa do vídeo e serão denominadas quadros.

A extração dos quadros foram realizados com o auxílio de uma função onde um vídeo e o número de amostras desejadas é fornecido, sendo extraídas imagens de dentro deste vídeo que são salvas em memória física para serem futuramente utilizadas no treino e validação.

Cada uma das amostras teve 5 quadros extraídos para serem utilizados como dado de entrada do treino e validação dos modelos, resultando em um total de 5000 amostras distribuídas igualmente entre as duas classes existentes.

B. Métodos

Com o intuito de se realizar o estudo de viabilidade proposto neste trabalho, modelos de identificação de eventos violentos foram desenvolvidos utilizando o *framework TensorFlow*.

Neste trabalho foi adotada a arquitetura proposta por [2], onde um modelo base é carregado através do processo de *transfer learning* e duas camadas são adicionas em sua saída, de tal forma que o desenho final é apresentado pela Figura 1.

O termo *transfer learning* se refere a utilização da capacidade de extração de características criada pelo treino de um modelo em um *dataset* maior e menos específico, na criação de um novo modelo utilizando um conjunto menor de dados [10].

Atualmente *convolutional neural networks* (CNN) se valem deste artifício para diminuir o tempo e complexidade do treino [10].

As CNN por sua vez são um tipo de rede neural artificial que utiliza uma ou mais camadas do tipo convolucional [11].

Estas camadas convolucionais têm por objetivo diminuir o tempo de treino ao reduzir a dimensionalidade dos dados utilizados ao agregar informações disponíveis [11] e desta forma gerando tensores de menor dimensão.

Tensores são objetos matemáticos gerados a partir da generalização de escalares, vetores e matrizes, de tal forma que podem representar qualquer um dos elementos anteriormente citados e outros com dimensões superiores [12].

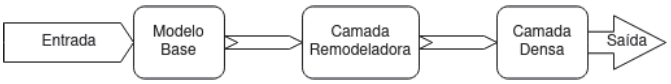


Fig. 1. Topologia dos modelos desenvolvidos.

No diagrama da Figura 1 a camada remodeladora se refere a utilização das camadas *Flatten* ou *GlobalAveragePooling2D* (GAP2D).

Estas camadas foram escolhidas durante o desenvolvimento deste estudo, uma vez que na descrição da arquitetura por [2] não evidencia a camada utilizada, e não é possível conectar diretamente a saída dos modelos pré-treinados em camadas densas.

A camada *Flatten* transforma um tensor de n dimensões em um tensor de duas, mantendo o nível de informação produzido pelo modelo base da etapa anterior.

No caso do GAP2D, uma média do tensor de saída do modelo base é utilizada, diminuindo o número de parâmetros para a camada subsequente e diminuindo o modelo.

Considerando que diversos estudos apontam que redes pré-treinadas utilizando o *dataset ImageNet* apresentam alta capacidade de detecção de ações [1], foram adotados modelos pré-treinados como modelo base, fazendo uso do *transfer learning*.

A camada final representada foi definida como uma camada do tipo densa com 2 neurônios de saída e a função de ativação *softmax*, tendo em vista que este é do tipo classificação multiclasse, onde a saída pode ser enquadrada como violenta ou não-violenta.

A definição da arquitetura como um problema multiclasse se deve aos dados utilizados no treinamento do modelo base, o conjunto *ImageNet*. Por se tratar de um conjunto com 1000 classes, toda a estrutura de pesos está alinhada para um problema do tipo multiclasse, que por sua vez requer o uso da função de ativação *softmax* ao invés da *sigmoid*.

Para o modelo base foram selecionados três modelos, o *InceptionV3* utilizado por [2], *ResNet50* utilizado por [8] e *MobileNetV2*.

Os dois primeiros modelos foram selecionados pois já foram utilizados em trabalhos anteriores e demonstraram capacidade de detecção de eventos violentos.

O último modelo foi selecionado a partir das informações disponibilizadas em [9], onde foi observado que o *MobileNetV2* apresenta um número reduzido de parâmetros e um baixo tempo de inferência.

Para o ajuste fino e treino dos modelos produzidos seguindo a arquitetura descrita na Figura 1 foi utilizada a função de perda $categorical\_crossentropy$ indicada para problemas de classificação multiclasse e a função de otimização *RMSProp*. Estas duas funções foram utilizadas em [1] e [8] e foram adotadas utilizando os hiperparâmetros propostos pelos mesmos e expostos na Tabela I.

Por fim, como métrica de treino, foi adotada a métrica de revocação. Esta métrica foi adotada com o intuito de se reduzir os falsos negativos, que representariam eventos de violência não detectados.

TABELA I
HIPERPARÂMETROS PARA O TREINO DOS MODELOS

	InceptionV3	ResNet50	MobileNetV2
Learning Rate	10^{-4}	10^{-4}	10^{-3}
Momentum	0.9	0	0
Batch	64	5	0
Epochs	5	5	5

Continuando para a etapa de treino, os quadros gerados a partir dos vídeos são carregados em memória no formato de 224x224 *pixels*. Após são convertidos em *Numpy Array* do tipo *float32* e tiveram seus valores normalizados ao serem divididos por 255.0.

A partir deste ponto os quadros são divididos em dois grupos de amostras, treino e teste, contendo respectivamente 4500 e 500 quadros ao utilizar a função $train\_test\_split$ do pacote *Sklearn*, com o parâmetro $test\_size$ igual a 0.1.

Os modelos depois de treinados foram avaliados sob a ótica das seguintes métricas:

- Tempo médio de inferência;
- Acurácia;
- Precisão;
- Revocação;
- *F1 - score*; e
- Tamanho do modelo.

Uma vez avaliados os modelos iniciais desenvolvidos utilizando o *TenssorFlow*, o que apresentava o melhor balanço em relação a tamanho do arquivo, tempo de execução e métricas foi selecionado para ser utilizado na criação dos modelos do *TensorFlow Lite*, que apresentam formato *FlatBuffer* e tem extensão do tipo .tflite [5].

A conversão do modelo pode ser realizada de duas formas distintas, utilizando a linha de comando do computador ou a *Application Programming Interface* (API) Python.

Neste trabalho foi utilizada a API Python pois permite maior controle dos parâmetros de otimização da compressão, permite incluir a compressão no mesmo *pipeline* de criação e por fim é a forma recomendada pela documentação da ferramenta [5].

Para a realização da compressão, foram utilizadas otimizações que reduzem o tamanho e latência do modelo, com nenhuma ou pouca perda de acurácia [5].

Dentre os tipos de otimização disponíveis, foi utilizada a quantização, onde existe a redução na precisão dos números utilizados como parâmetros dos modelos, que por padrão são do tipo *float32* [6].

O motivo da adoção deste tipo de otimização é devido ao fato que modelos prontos podem ser otimizados, sem a necessidade de se interferir na arquitetura e treino já existentes.

A alteração da precisão dos parâmetros resulta em modelos de menor tamanho e tempo de execução menor, além de requererem menores quantidades de memória de armazenamento e *RAM* do equipamento utilizado no processo de inferência [6].

Dentro dos modos de quantização disponíveis foram utilizados os seguintes:

- Quantização de faixa dinâmica; e
- Quantização *Float16*.

Na quantização de faixa dinâmica os pesos são convertidos de ponto flutuante para inteiros com 8 *bits* de precisão, contudo durante a etapa de inferência os pesos são convertidos novamente em ponto flutuante [7].

No caso da Quantização *Float16* os parâmetros são convertidos de *float32* para *float16*, causando grande redução no modelo com baixa perda de acurácia [7].

Desta forma, o modelo criado utilizando o *TensorFlow* na etapa anterior deste trabalho foi convertido em três modelos compactados, um deles sem nenhuma otimização e dois com as descritas anteriormente.

A conversão do modelo original para o formato .tflite foi realizado com a utilização do trecho de código disponível no Quadro 1.

No caso da quantização de faixa dinâmica a propriedade *optimizations* do objeto *converter* é acessada e recebe o valor tf.lite.Optimize.DEFAULT.

Já no caso da quantização *Float16* além da mesma configuração utilizada para a faixa dinâmica, a propriedade $target\_spec.supported\_types$ recebe o valor tf.float16.

```
#importando o TensorFlow
import tensorflow as tf

# criacao de instancia do conversor do modelo
converter = tf.lite.TFLiteConverter.from_keras_model
    (model)

#criacao do modelo .tflite
tflite_model = converter.convert()

# criacao do caminho e nome do arquivo
name_with_extension = name + ".tflite"
full_path = os.path.join(path, name_with_extension)

# salvando modelo compactado em disco
with open(full_path, 'wb') as f:
    f.write(tflite_model)
```

QUADRO 1. Código para conversão do modelo em TensorFlow Lite

Uma vez criados os modelos compactados, foi realizada uma etapa de inferência, extraindo as mesmas métricas utilizadas para avaliação dos modelos gerados utilizando o *TensorFlow*.

A inferência no modelo do *TensorFlow Lite* é realizada com o auxílio de um interpretador e o código utilizado está disponível no Quadro 2.

```
# Carregando modelo em memoria
interpreter = tf.lite.Interpreter(model_path="")
interpreter.allocate_tensors()

# Determinando formato de entrada e saida
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Realizando inferencia
interpreter.invoke()
output_data=interpreter.get_tensor(output_details)
```

QUADRO 2. Código para inferência com modelo TensorFlow Lite

C. Tecnologias

Para execução dos experimentos utilizados neste trabalho foram utilizados o *Google Colab* com 12 GB de RAM, sem a utilização de aceleradores gráficos e os seguintes programas:

- Python 3.7.13;
- TensorFlow 2.8.2;
- Numpy 1.21.6;
- OpenCV 4.6.0;
- Matplotlib 3.2.2;
- Sklearn 1.0.2;
- Pacotes do Python: os; zipfile, time e gc.

TABELA II
MÉTRICAS DOS MODELOS CRIADOS EM TENSORFLOW

	InceptionV3 e GAP2D	InceptionV3 e Flatten	ResNet50 e GAP2D	ResNet50 e Flatten	MobileNetV2 e GAP2D	MobileNetV2 e Flatten
Tempo médio de inferência(ms)	166.7	166.9	286	166	34.238	38.198
Acurácia(%)	90.8	98.2	83	56.2	95.6	99.2
Precisão(%)	89.855	98.491	92.891	54.845	94.853	99.248
Revocação(%)	93.233	98.120	73.684	100	96.992	99.248
F1 - score(%)	91.513	98.305	82.180	70.839	95.911	99.248
Tamanho do modelo(MB)	84.042	84.792	90.498	91.247	9.083	9.551

TABELA III
MÉTRICAS DOS MODELOS CRIADOS PELO TENSORFLOW LITE

	Sem otimização	Quantização Dinâmica	Quantização Float16
Tempo médio de inferência(ms)	23.529	50.375	24.427
Acurácia(%)	99.2	98.6	99.2
Precisão(%)	99.248	98.502	99.248
Revocação(%)	99.248	98.872	99.248
F1 - score(%)	99.248	98.687	99.248
Tamanho do modelo(MB)	8.928	2.506	4.495

II. RESULTADOS E DISCUSSÕES

Os modelos criados inicialmente utilizando o *TensorFlow* foram avaliados e os valores das métricas obtidas podem ser observados na Tabela II, demonstrando que o modelo criado a partir do modelo base *MobileNetV2* com a camada remodeladora *Flatten* apresentaram uma combinação balanceada de tamanho, tempo de inferência e métricas, sendo este o selecionado para a etapa de compressão.

Ainda sobre os modelos gerados, podemos observar que o modelo *ResNet50* com a camada remodeladora *Flatten* apresenta características de *overfit* tendo em vista que a métrica utilizada como função de perda apresenta valor de 100%, mas a acurácia do modelo se aproxima de 50%, o que seria o mesmo de um evento totalmente aleatório para os casos de teste.

Observando mais a fundo os valores da função de perda durante o treinamento disponível na Figura 2, podemos observar que na última iteração de treino o valor da perda para a massa de teste, representada por val_loss apresenta um incremento abrupto, sendo indicativo da degradação da capacidade do modelo em generalizar e indicando o *overfit*.

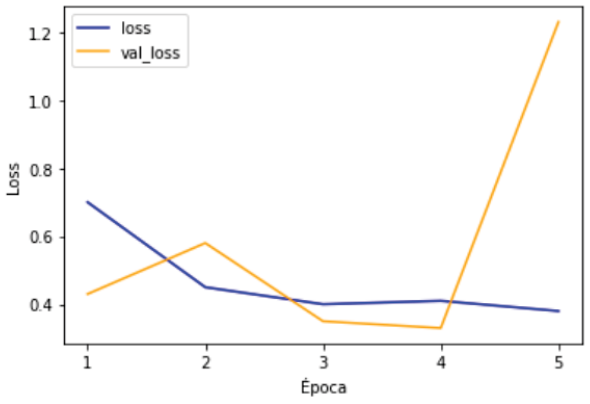


Fig. 2. Gráfico da função de perda durante o treino da rede ResNet50 com Flatten.

Outra tendência que pode der observada nos modelos *InceptionV3* e *MobileNetV2* é a de que os modelos criados utilizando a camada *Flatten* apresentam métricas de acurácia, precisão, revocação e *F1 - score* superiores a de suas contra partes que utilizaram o GAP2D, com o custo de um incremento no tamanho do modelo.

Desta forma, podemos observar que camadas do como a GAP2D que agregam valores são a melhor opção para o desenvolvimento de modelos com menor tamanho.

Continuando com a análise dos modelos gerados no *TensorFlow Lite*, e observando as métricas disponíveis na Tabela III pode ser observado que a utilização da compressão do modelo original gerado em *TensorFlow* sem otimização não afeta as métricas do modelo, além de reduzir o tamanho em 6.5% e o tempo de inferência em 15 ms.

Seguindo a análise para o modelo com quantização dinâmica, podemos observar que ocorre a perda na capacidade de detecção do modelo, além de um aumento expressivo no tempo de inferência, sendo o mesmo superior ao do modelo original gerado. Contudo observando o tamanho do arquivo, existe uma redução de pouco mais de 73%, o que se apresenta como sendo o ideal para equipamentos com baixa capacidade de memória.

Por fim, a quantização *Float16*, na qual as métricas se mantém idênticas as observadas no modelo sem compressão e observa-se a diminuição do tempo de inferência de pouco mais de 13 ms em comparação do modelo original. Observando o tamanho do modelo, percebe-se a diminuição de mais de 50%, o que aponta este modelo como o com o melhor balanço entre fator de redução do arquivo, perda de performance e capacidade de detecção.

Desta forma podemos concluir que a compressão de modelos de detecção de eventos violentos com a utilização do *TensorFlow Lite* é uma técnica válida para permitir que estes modelos sejam utilizados em equipamentos com menor capacidade de processamento e memória, com baixa ou nenhuma perda das capacidades de detecção de eventos.

III. Trabalhos Futuros

A partir das discussões e importância do tema levantado neste trabalho, pode-se observar que muito ainda pode ser desenvolvido neste segmento.

Desta forma como proposta para trabalhos futuros pode-se valer o estudo em:

- Criação de modelos que levem em conta a temporalidade dos eventos dos vídeos;
- Comparação entre modelos pré-treinados com dados distintos e sua capacidade de detecção de eventos violentos; e
- Execução e métricas de modelos .tflite em diversos equipamentos de baixo custo.

Referências

[1] S. Sudhakaran, e O. Lanz.Learning to Detect Violent Videos using Convolutional Long Short-Term Memory, IEEE, 2017.

[2] A. Mumtaz, A. B. Sargano, e Z. Habib. Violence Detection in Surveillance Videos with Deep Network using Transfer Learning. EECS, 2018.

[3] IHS Markit's Top Video Surveillance Trends For 2018. Disponível em: https://www.ihs.com/info/1217/top-video-surveillance-trends-2018.html. Acessado 10 Agosto 2022.

[4] Y. Shrief. Violence detection-Hockey Fight-CNN+LSTM. Disponível em: https://www.kaggle.com/code/yassershrief/violence-detection-hockey-fight-cnn-lstm/data, Acessado 16 Agosto 2022.

[5] TensorFlow. Conversor do TensorFlow Lite. Disponível em: https://www.tensorflow.org/lite/convert/index?hl=pt-br. Acessado 17 Agosto 2022.

[6] TensorFlow. Otimização do modelo. Disponível em: https://www.tensorflow.org/lite/performance/model_optimization?hl=pt-br. Acessado 17 Agosto 2022.

[7] TensorFlow. Quantização pós-treinamento. Disponível em: https://www.tensorflow.org/lite/performance/post_training_-quantization?hl=pt-br. Acessado 17 Agosto 2022.

[8] M. B. Patel. Real-Time Violence Detection Using CNN-LSTM, Arxiv, 2021.

[9] Keras. Keras Applications. Disponível em : https://keras.io/api/applications/. Acessado 12 Outubro 2022.

[10] C.C. Aggarwal. "Neural Networks and Deep Learning: A Textbook", Springer International Publishing AG, 2018, pp.751-752, 813-816. DOI: 10.1007/978-3-319-94463-0

[11] S. Skansi. "Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence", Springer International Publishing AG, 2018, pp.291-292. DOI: 10.1007/978-3-319-73004-2

[12] T. Rowland e E. Weisstein W. "Tensor". Disponível em: https://mathworld.wolfram.com/Tensor.html. Acessado 27 Outubro 2022.