Relatório do Projeto de Classificação de Imagens

Grupo Formado por:
- Joan Vitor
- Matheus Morais
- Renilton Ribeiro
- Yooh Brito

1. Introdução

Nos últimos anos, o avanço de tecnologias que utilizam inteligência artificial, IA, trouxe inúmeras melhorias no cotidiano da sociedade em todo o mundo. Melhorias que abrangem todas as áreas, desde encontrar anúncios na web para um usuários comum até localizar planetas e galáxias a milhões de anos-luz do nosso planeta. Um dos principais avanços é a utilização da IA para classificar imagens, como é o caso do ReCaptcha, algoritmo do Google responsável por evitar que robôs prejudiquem algum sistema. Atualmente, a classificação de imagens é largamente utilizada na Medicina para identificar doenças, através de algoritmos capazes de classificar em uma imagem, tumores ou outras anomalias no corpo humano. Outra área, em que a classificação é abrangente, é a de segurança, como é o caso de aeroportos que são capazes de identificar objetos letais em bagagem de tripulantes. Sendo assim, o atual projeto visa ser capaz de classificar se em uma imagem contém um objeto específico, no caso deste projeto, chapéus, utilizando o reconhecimento para analisar se a foto atende aos requisitos de um passaporte.

O desenvolvimento de algoritmos avançados como Machine Learning e Deep Learning, possibilitaram a IA, resultados precisos de análises de base de dados complexas, como o mapeamento de rotas de trânsitos via internet, elaboração de relatórios financeiros de uma determinada empresa, identificação de criminosos por câmeras de segurança, entre outras utilidades. Diante disso, no presente projeto, para detecção de imagens, foi utilizado a rede Neural YOLO - You Only Look Once, em conjunto ao Darknet, com o objeto de classificar a presença de objetos do tipo chapéu, em uma imagem de passaporte.

Diante disso, o presente projeto, é subdividido em:
- Fundamentação Teórica:
- Materiais e Métodos: Documentação da tecnologias e do processo de desenvolvimento da aplicação, além da base de dados utilizada.
- Considerações Finais: Resumo dos resultados encontrados após o desenvolvimento da aplicação
- Referências: Listagem do conteúdo utilizado como fundamentação durante o desenvolvimento.

2. Fundamentação Teórica

Nesta seção, é documentada toda a teoria do desenvolvimento da aplicação.

2.1. Inteligencia Artificial

A tecnologia, a Inteligência Artificial (IA) é a inteligência demonstrada por máquinas ao executar tarefas complexas associadas a seres inteligentes, além de também ser um campo de estudo acadêmico. Onde o principal objetivo é de executar funções de modo autônomo que, caso um ser humano fosse executar, seriam consideradas inteligentes.

É um conceito amplo, e que recebe tantas definições quanto significados diferentes à palavra inteligência. Podemos pensar em algumas características básicas desses sistemas, como a capacidade de raciocínio (aplicar regras lógicas a um conjunto de dados disponíveis para chegar a uma conclusão), aprendizagem (aprender com os erros e acertos de forma que no futuro agirá de maneira mais eficaz), reconhecer padrões (tanto padrões visuais e sensoriais, como também padrões de comportamento) e inferência (capacidade de conseguir aplicar o raciocínio nas situações do nosso cotidiano).

Dentre áreas em que IA é utilizada, estão:

- Medicina
- Segurança
- Astrologia
- Mobilidade Urbana
- Ações e bolsa de valores
- Criptografia

2.2. Machine Learning

Machine Learning, em sua tradução, aprendizado de máquina é um sistema que pode modificar seu comportamento autonomamente tendo como base a sua própria experiência — o treinamento que abordamos anteriormente, onde a interferência humana é mínima. Em outras palavras, é um sistema capaz de, a partir de uma base de dados, “aprender” a processar dados e “devolver” uma resposta para a análise feita. Portanto, o ML é um método de análise de dados que automatiza o desenvolvimento de modelos analíticos, por meio de algoritmos que aprendem interativamente a partir de dados, permitindo assim que os computadores encontrem insights ocultos sem serem explicitamente programados para procurar algo específico.

Dentre os seus métodos de aprendizagem, estão:

- **Aprendizado supervisionado**: consiste em exemplos rotulados. O algoritmo de aprendizagem recebe um conjunto de entradas junto com as saídas corretas correspondentes, além disso, o algoritmo é capaz de “aprender” ao comparar a saída real com as saídas corretas para encontrar erros. Em seguida, ele modifica o modelo de acordo.

- **Aprendizado não supervisionado**: utilizado contra dados que não possuem rótulos históricos. Ou seja, o sistema não sabe a “resposta certa” nesse caso. O algoritmo deve descobrir o que está sendo mostrado e o objetivo é explorar os dados e assim encontrar alguma estrutura neles. O aprendizado não supervisionado funciona bem em dados transacionais.

- **aprendizado semisupervisionado**: geralmente é utilizado para as mesmas aplicações que o aprendizado supervisionado, porém ele pode usar tanto dados rotulados quanto não marcados para o treinamento – normalmente uma pequena quantidade de dados rotulados com uma grande quantidade de dados não rotulados (pois os dados não rotulados são mais baratos e necessitam de menos esforço para serem adquiridos). Esse tipo de aprendizagem pode ser utilizado com métodos como a classificação, regressão e previsão. O aprendizado semisupervisionado é útil quando o custo associado à rotulagem é demasiado alto para permitir um processo de treinamento totalmente rotulado. Os primeiros exemplos disso incluem a identificação do rosto de uma pessoa em uma webcam.

2.3. Deep Learning

A aprendizagem profunda, ou simplesmente Deep Learning, é parte da família mais abrangente de métodos de ML baseados na aprendizagem de representações de dados. Uma observação (por exemplo, uma imagem), pode ser representada de várias maneiras, tais como um vetor de valores de intensidade por pixel, ou de uma forma mais abstrata como um conjunto de arestas, regiões com um formato particular, etc. Algumas representações são melhores do que outras para simplificar a tarefa de aprendizagem (por exemplo, reconhecimento facial ou reconhecimento de expressões faciais). Uma das promessas da aprendizagem profunda é a substituição de características feitas manualmente por algoritmos eficientes para a aprendizagem de características supervisionada ou semissupervisionada e extração hierárquica de características.

Os primeiros registros científicos da tentativa de reprodução de um neurônio artificial datam da década de 50, onde modelos computacionais foram desenvolvidos, porém

pouco explorados devido à falta de poder de processamento dos computadores da época. Com o desenvolvimento da era digital, os computadores tornaram-se mais poderosos do ponto de vista de processamento, o que permitiu avanços significativos na área. Como uma Rede Neural é um paradigma conexionista ponderados por pesos, ou seja, a capacidade de desenvolver modelos inteligentes está nas conexões de uma quantidade significativa de neurônios artificiais (e não nos neurônios em si), as operações aritméticas crescem de forma exponencial no tocante à Redes Profundas.

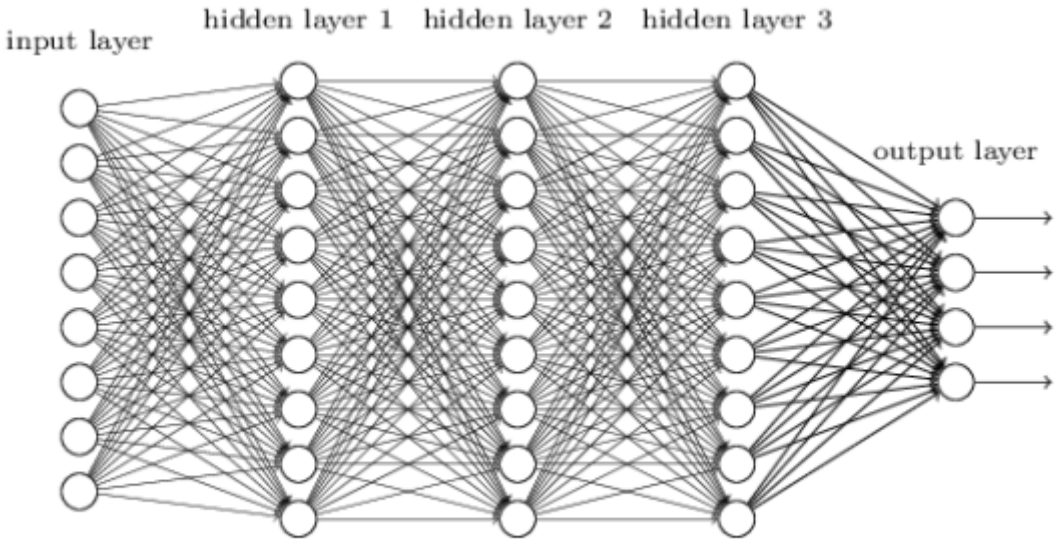


Figura 1: Paradigma conexionista de uma Rede Neural Artificial

Dentre as principais aplicabilidades do Deep Learning estão:

- Processamento de Imagem;
- Processamento de Linguagem Natural
- Aplicações em Medicina:
    - Aplicações de Reconhecimento de Imagens
    - Câncer de Mama
    - Doença de Alzheimer
    - Diagnóstico Cardiovascular
    - Câncer de Pele
    - Derrame Cerebral
    - Aplicações em Desenvolvimento de Medicamentos
- Genômica

2.4. Redes Neurais

Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência. Uma grande rede neural artificial pode ter

centenas ou milhares de unidades de processamento; já o cérebro de um mamífero pode ter muitos bilhões de neurônios.

2.3.1 Características Gerais das Redes Neurais

Uma rede neural artificial é composta por várias unidades de processamento, cujo funcionamento é bastante simples. Essas unidades, geralmente, são conectadas por canais de comunicação que estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede.

A operação de uma unidade de processamento, proposta por McCullock e Pitts em 1943, pode ser resumida da seguinte maneira:

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um número, ou peso, que indica a sua influência na saída da unidade;
- É feita a soma ponderada dos sinais que produz um nível de atividade;
- Se este nível de atividade exceder um certo limite (threshold) a unidade produz uma determinada resposta de saída.

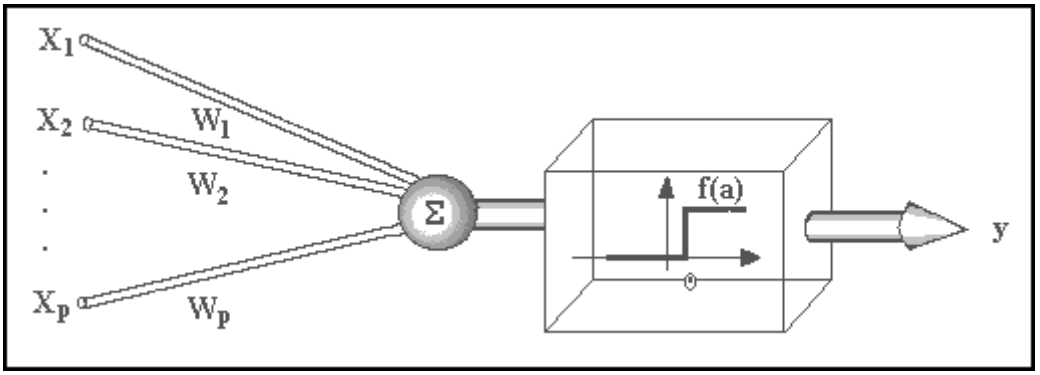


Figura 2: Esquema de unidade McCullock - Pitts.

Suponha que tenhamos p sinais de entrada X1, X2, ..., Xp e pesos w1, w2, ..., wp e limitador t; com sinais assumindo valores booleanos (0 ou 1) e pesos valores reais.

Neste modelo, o nível de atividade a é dado por:

a = w1X1 + w2X2 + ... + wpXp

A saída y é dada por:

y = 1, se a >= t ou
y = 0, se a < t.

A maioria dos modelos de redes neurais possui alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados, em outras palavras, elas aprendem através de exemplos. Arquiteturas neurais são tipicamente organizadas em camadas, com unidades que podem estar conectadas às unidades da camada posterior.

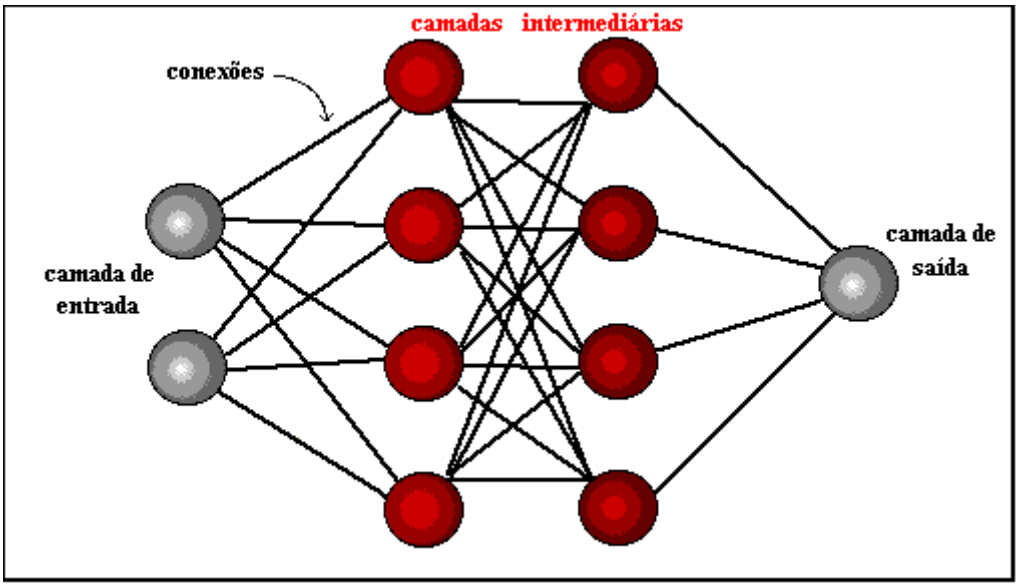


Figura 3: Organização em camadas.

Usualmente as camadas são classificadas em três grupos:

- **Camada de Entrada**: onde os padrões são apresentados à rede;
- **Camadas Intermediárias ou Escondidas**: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características;
- **Camada de Saída**: onde o resultado final é concluído e apresentado.

Uma rede neural é especificada, principalmente pela sua topologia, pelas características dos nós e pelas regras de treinamento. A seguir, serão analisados os processos de aprendizado.

2.3.2. Processos de Aprendizado

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e com isso melhorar seu desempenho. Isso é feito através de um processo iterativo de ajustes aplicado a seus pesos, o treinamento. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados.

Outro fator importante é a maneira pela qual uma rede neural se relaciona com o ambiente. Nesse contexto existem os seguintes paradigmas de aprendizado:

- **Aprendizado Supervisionado**, quando é utilizado um agente externo que indica à rede a resposta desejada para o padrão de entrada;
- **Aprendizado Não Supervisionado** (auto-organização), quando não existe uma agente externo indicando a resposta desejada para os padrões de entrada;
- **Reforço**, quando um crítico externo avalia a resposta fornecida pela rede.

Denomina-se ciclo uma apresentação de todos os N pares (entrada e saída) do conjunto de treinamento no processo de aprendizado. A correção dos pesos num ciclo pode ser executado de dois modos:

1. **Modo Padrão**: A correção dos pesos acontece a cada apresentação à rede de um exemplo do conjunto de treinamento. Cada correção de pesos baseia-se somente no erro do exemplo apresentado naquela iteração. Assim, em cada ciclo ocorrem N correções.
2. **Modo Batch**: Apenas uma correção é feita por ciclo. Todos os exemplos do conjunto de treinamento são apresentados à rede, seu erro médio é calculado e a partir deste erro fazem-se as correções dos pesos.

2.5. Yolo e Darknet

A detecção de objetos em imagens é uma das aplicações de machine learning que tem evoluído bastante nos últimos anos. Conseguir ensinar um programa de computador sobre o conteúdo de imagens abriu espaço para diversas aplicações, desde melhores recomendações no seu feed do Instagram, até o desenvolvimento de veículos autônomos.

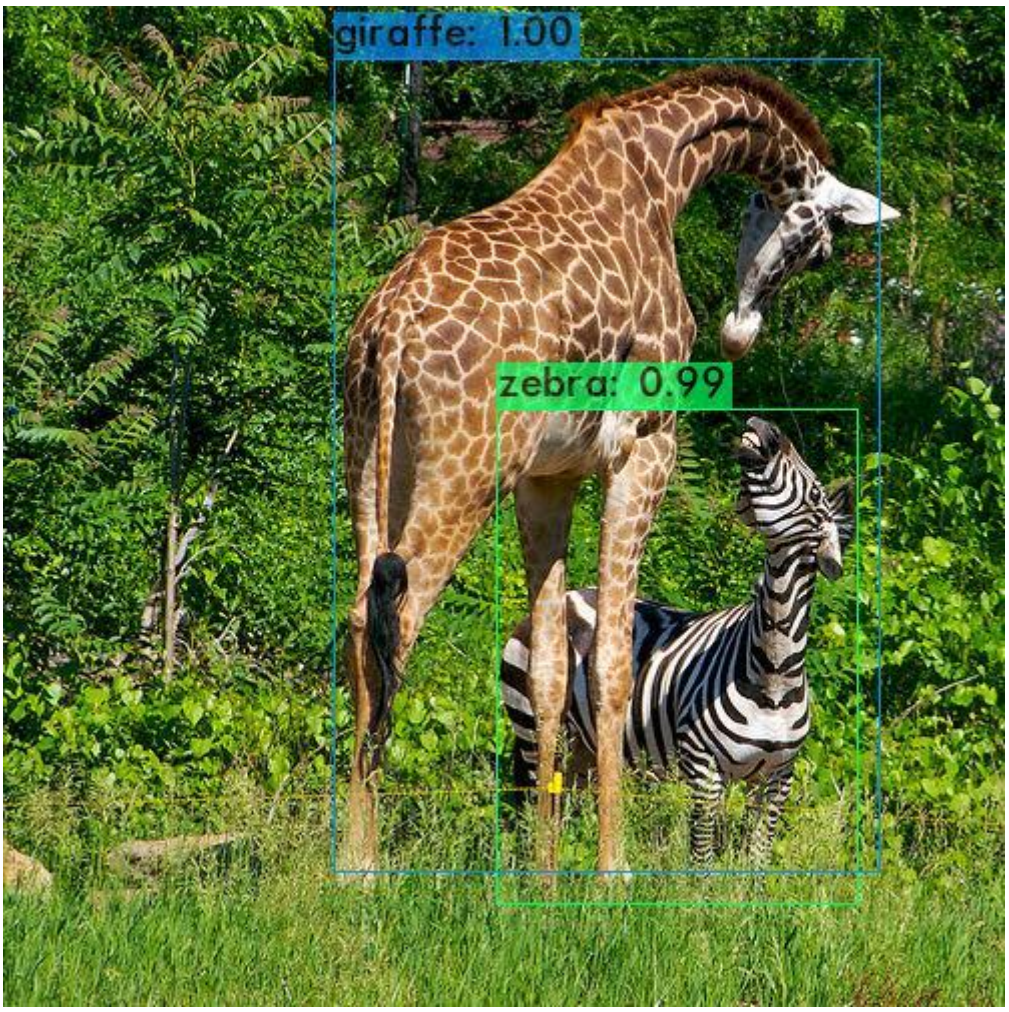
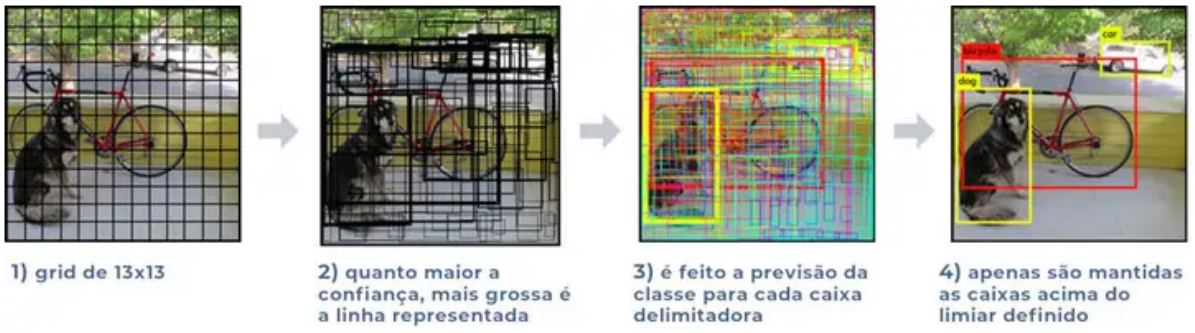


Figura 4: Detecção de animais em uma imagem.

YOLO é um método de detecção de objetos de passada única (single pass) que utiliza uma rede neural convolucional como extrator de características (features). Diferente de algoritmos anteriores de detecção de objetos, como R-CNN ou Faster R-CNN, ele apenas precisa olhar pela imagem uma única vez para enviar para a rede neural. Por isso ele recebe esse nome (You Only Look Once – “Você só olha uma vez”). E devido a essa característica, o YOLO foi capaz de conseguir uma velocidade na detecção muito maior do que as técnicas concorrentes, sem perder em acurácia.

O YOLO trata a detecção de objetos como um simples problema de regressão.



1. Primeiro, o algoritmo divide a imagem em um grid de S x S células. Como exemplo usaremos 13×13, porém esse tamanho pode mudar. Para as versões mais recentes, por exemplo, tem-se preferido utilizar um grid de 19×19.
2. Cada uma dessas células é responsável por fazer a predição de 5 caixas delimitadoras (B), para caso haja mais de um objeto naquela célula. Também é retornada a pontuação de confiança que nos diz o quanto de certeza ele tem que aquela caixa delimitadora contenha um objeto.
3. Para cada caixa, a célula também faz a previsão de uma classe. Isso funciona como se fosse um classificador: é fornecido um valor de probabilidade para cada uma das classes possíveis. O valor de confiança para a caixa delimitadora e a predição da classe são combinados em uma pontuação final, que vai nos dizer a probabilidade dessa caixa conter um objeto específico.
4. Nesse caso o grid é 13×13, o que no final resulta em 169 células. Para cada uma dessas células são detectados 5 caixas delimitadoras, o que resulta em 845 no total.
5. Acontece que a maioria dessas caixas terá um valor de confiança extremamente baixo, por isso geralmente se considera apenas as caixas cuja pontuação final seja 30% ou mais. Esse valor de 30% é o limiar, chamado de threshold, e ele pode ser alterado dependendo do quão preciso você quer que o detector seja.

Para o seu funcionamento o YOLO utiliza uma rede neural profunda, cuja arquitetura é chamada de Darknet, que é o mesmo nome do framework utilizado para implementar o detector.Esse framework foi desenvolvido pelo próprio criador do YOLO, Joseph Redmon.sendo um open source e escrito na linguagem C, também possui suporte para GPU.

Figura 5: Logo do Darknet

3. Materiais e Métodos

A presente seção tem como objetivo documentar todo o trabalho realizado no projeto, evidenciando as etapas que foram percorridas, ilustrar os tipos de dados utilizados, além de listar as ferramentas/frameworks utilizadas durante o desenvolvimento, para isso, foi-se necessário dividi-lo em dois processos: Criação da Base de Dados/Dataset e Treinamento da Rede Neural Yolo.

3.1. Criação da Base de Dados/Dataset

Para a configuração da base de dados utilizada durante o desenvolvimento desse projeto, foi necessário:

- Imagens catalogadas com referência de Bounding Box dos objetos a serem detectados.
- Arquivo .txt, chamado obj.names, com os nomes das classes dos objetos utilizados.
- Arquivo .txt, chamado obj.data, com as imagens e as referências da Bounding Box.
- Arquivo de configuração da estrutura da rede neural YOLO.[
- Arquivos Train e Test, contendo os casos de treinamento e teste para a rede neural.

As imagens utilizadas como base foram fornecidas a partir do banco de imagens **Open Images Dataset,** o qual dispõe de mais de 9 milhões de imagens catalogadas. As classes definidas no projeto são:

- Chapéu,
- Chapéu de Cowboy

- Chapéu de Sol
- Touca de Natação
- Hijab, conjunto de vestimentas preconizado pela doutrina islâmica, esse banco criado por nós.

Para o download delas, foi utilizada a ferramenta OID, versão 4.0, que seguiu as seguintes etapas:

Etapa 1: Clone do OIDv4 ToolKit

Para esta etapa, é necessário a utilização do comando *git clone.*

```
!git clone https://github.com/EscVM/OIDv4_ToolKit.git
```

Etapa 2: Acesso ao diretório da biblioteca

Em seguida, deve-se acessar a pasta do OIDv4.

```
cd OIDv4_ToolKit/
```

Etapa 3: Instalação das bibliotecas utilizadas pelo OIDv4_Toolkit

Dentro da pasta da ferramenta, existe um arquivo denominado r*equirements.txt*, por meio deste é possível realizar o download de todas as bibliotecas necessárias para o funcionamento do OIDv4**.**

```
!pip3 install -r requirements.txt
```

Etapa 4: Download das Imagens

Para o download das imagens, são definidas as seguintes classes de imagens:

- Hat
- Cowboy Hat
- Sun Hat
- Swim Cap

Em seguida, executa-se o arquivo *main.py* passando o nome das classes como parâmetro, primeiro são baixadas as imagens de treinamento, com um total de 1000 imagens, em seguida são baixadas as imagens de teste, com 200 imagens.

```
!python main.py downloader --classes Hat Cowboy_hat Sun_hat Swim_cap --type_csv train --limit 1000 --multiclasses 1
```

```
!python main.py downloader --classes Hat Cowboy_hat Sun_hat Swim_cap --type_csv test --limit 200 --multiclasses 1
```

Obs.: Outros parâmetros definidos:

- **train/test:** Indicação se as imagens são de treinamento ou teste
- **multiclasses:** As classes deverão ser salvas no mesmo pacote

Para cada imagem, são definidos um arquivo .jpg e um txt, contendo as informações do bounding box da imagem.

Etapa 5: Conversão dos arquivos de anotação

Antes de executar a rede neural, é necessário converter o formato do arquivo txt da imagem para o padrão que o YOLO utiliza. Para isso, primeiramente, utiliza-se o comando *!echo* que salva dentro de classes.txt as classes utilizadas:

```
!echo -e 'Hat\nCowboy hat\nSun hat\nSwim cap\nHijab' > classes.txt
```

Em seguida, conectamos ao drive e baixamos o arquivo *treinamentoYOLO,* e então após descompactá-lo, executamos o *converter_annotation.py:*

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
!unzip /content/gdrive/MyDrive/YOLO/recursos/TreinamentoYOLO.zip -d /content/
```

```
!python converter_annotations.py
```

Obs.: Formato compatível com o YOLO:

- <classe-id> <x> <y> <largura> <altura>

Onde:

- <classe-id>, o identificador do arquivo
- <x>, coordenada x do ponto inicial
- <y>, coordenada y do ponto inicial
- <largura>, largura_do_objeto/largura_da_imagem
- <altura>, altura_do_objeto/altura_da_imagem

O *converter_annotations.py* converte os arquivos de treinamento e teste para o padrão yolo.

Etapa 6: Compactação do Dataset

Dentro da pasta train, executa-se o comando *!zip* para compactar o dataset de treinamento.

```
!zip -r ../../../obj.zip obj -x obj/Label/*
```

O mesmo processo se repete para a pasta test, ou seja, o dataset de teste.

```
!zip -r ../../../../valid.zip valid -x valid/Label/*
```

Move-se os arquivos para um diretório de referência.

```
!cp ./obj.zip /content/gdrive/MyDrive/YOLO/recursos
```

```
!cp ./valid.zip /content/gdrive/MyDrive/YOLO/recursos
```

Etapa 7: Edição dos arquivos de Configuração

Download do Darknet:

```
!git clone https://github.com/AlexeyAB/darknet
```

Compilação do Darknet:

```
!make
```

Modificação do arquivo .cfg:

```
[convolutional]
size=1
stride=1
pad=1
filters=30
activation=linear


[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 7
classes=5
```

```
# Training
#width=512
#height=512
width=224
height=224
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.0013
burn_in=1000
# 2000 * classes = 2000 * 5 = 10000
max_batches = 10000
policy=steps
# 80% e 90% dos batches
steps=8000,9000
```

Obs.: Modificações:

- width=224 e heigth=224, para melhorar o desempenho
- max_batches, definido para 10000, utiliza o seguinte cálculo: 2000*(Número de Classes).
- Steps: Porcentagem de batches que serão utilizados
- classes: Número de classes utilizadas
- filters: Números de filtros, por recomendação da documentação, utilizamos o cálculo:
    - (Número de classes + 5) * 3.

Criação das classes obj.names e obj.data:

```
!touch obj.names
!touch obj.data
```

Mover os arquivos obj.names e obj.data para a pasta de referência:

```
!cp obj.names /content/gdrive/MyDrive/YOLO/recursos/obj.names
!cp obj.data /content/gdrive/MyDrive/YOLO/recursos/obj.data
```

Obs.: Alterações nos arquivos:

- obj.names
- obj.data

Etapa 8: Gerando os arquivos Train e Test

Descompilar os arquivos salvos na pasta de referência do treinamento e de teste do dataset:

```
!unzip obj.zip -d ./data

!unzip /content/valid.zip -d ./data
```

Executar os códigos gerar_train e gerar_test para converter os arquivos txt de train e test para o padrão YOLO:

```
!python /content/TreinamentoYOLO/gerar_train.py

!python /content/TreinamentoYOLO/gerar_test.py
```

Mover o train.txt e o test.txt convertidos para a pasta de referencia:

```
!cp train.txt /content/gdrive/MyDrive/YOLO/recursos/train.txt

!cp test.txt /content/gdrive/MyDrive/YOLO/recursos/test.txt
```

3.2. Treinamento do Yolo:

Para realizar o treinamento da rede neural, é necessário utilizar a GPU no processamento, com o objetivo de melhorar o desempenho, para é necessario o uso da biblioteca *tensorflow*:

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
print(device_name)
```

Etapa 1: Conectando ao Google Drive

Conecta ao google através da biblioteca *google colab*:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Cria um caminho para a pasta:

```
!ln -s /content/gdrive/MyDrive/YOLO/ /yolo
```

Etapa 2: Download do Darknet

Clona o repositório do Darknet

```
!git clone https://github.com/AlexeyAB/darknet
```

Etapa 3: Compilando a biblioteca

Executa instruções para o colab utilizar a GPU:

```
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

Compila o darknet:

```
!make
```

Etapa 4: Preparando o YOLO

Função para plotar na tela o resultado/imagem:

```
import cv2
import matplotlib.pyplot as plt

def mostrar(caminho):
  img = cv2.imread(caminho)
  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.show()
```

Etapa 5: Enviando o dataset customizado para o Colab

Download dos pesos, referência, do modelo pré-treinado:

```
!unzip /yolo/recursos/obj.zip -d ./data/
```

```
!unzip /yolo/recursos/valid.zip -d ./data/
```

Move o conjunto de imagens de validação, ou teste, junto com o txt para dentro da pasta do darknet:

```
!cp /yolo/recursos/yolov4_custom.cfg ./cfg
!cp /yolo/recursos/obj.names ./data
!cp /yolo/recursos/obj.data ./data
!cp /yolo/recursos/train.txt ./data
!cp /yolo/recursos/test.txt ./data
```

Etapa 6: Baixando os pesos pré-treinandos das camadas convolucionais

Download dos arquivos com os pesos - Características básicas sobre a imagem.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

Obs.: Essa etapa é necessária para melhorar a performance, caso contrário, a rede neural deverá ser treinada novamente. No geral, é uma transferência de aprendizado

Etapa 7: Treinamento do Modelo

Execução do darknet, passando como parâmetros:

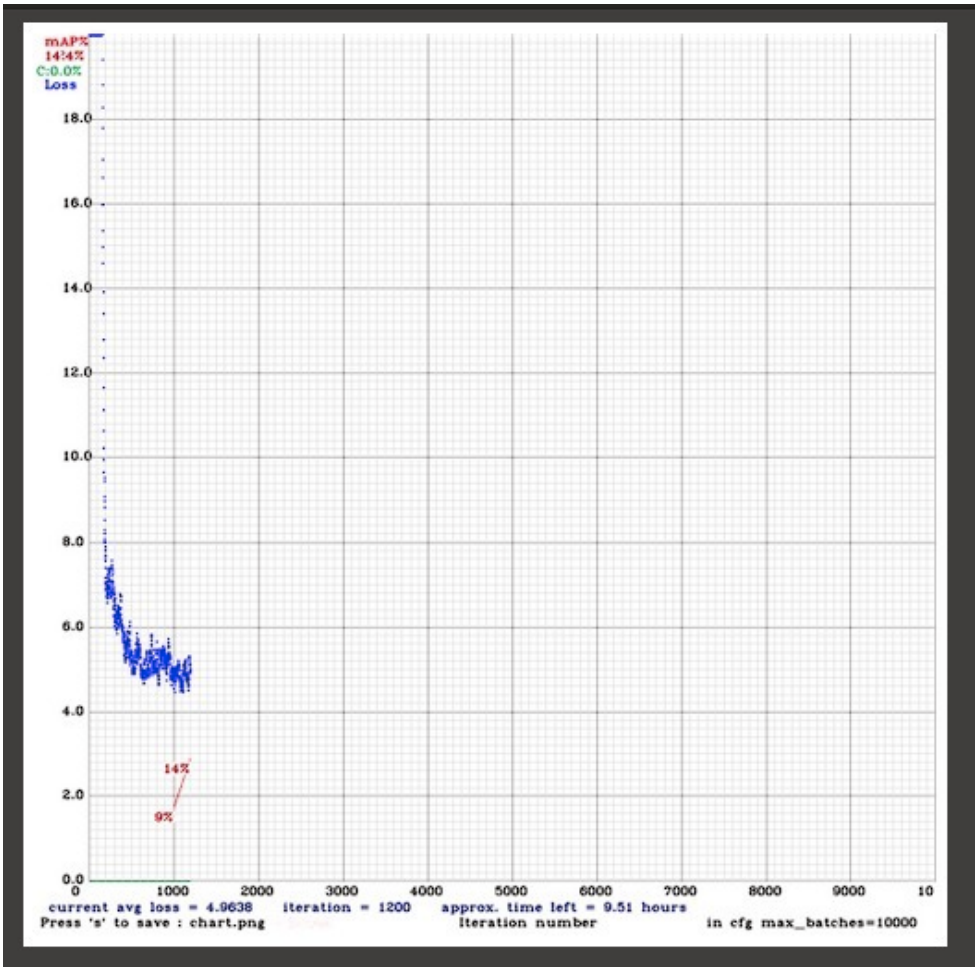
- train = comando que indica um treinamento
- yolov4_custom = configurações customizadas do yolo
- obj.data = dataset personalizado.
- yolov4_custom_best = pesos pré-definidos

```
!./darknet detector map data/obj.data cfg/yolov4_custom.cfg /content/gdrive/MyDrive/YOLO/recursos/yolov4_custom_best.weights
```

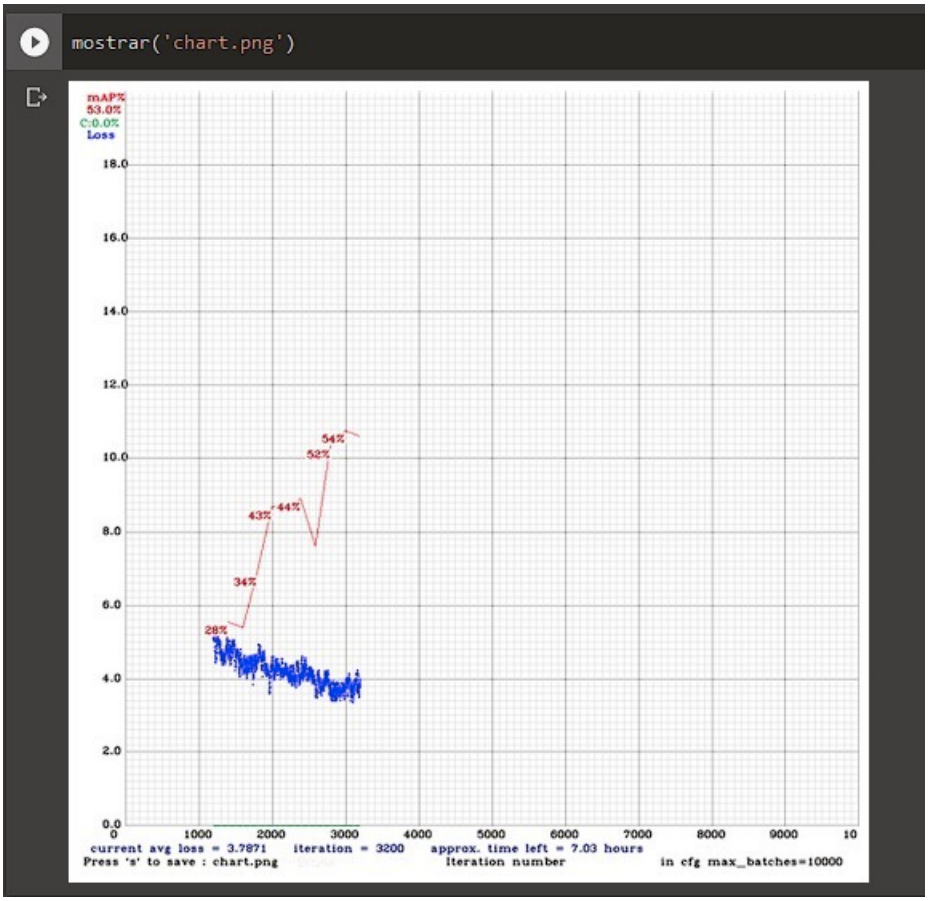
Mostrar o gráfico com as porcentagens de erro:

```
mostrar('chart.png')
```

1000 Batches



3000 Batches



Obs: À medida que mais casos são executados por mais tempo, a taxa de erro diminui.

Verificação das métricas de avaliação para obter a média de acerto da rede neural:

```
!./darknet detector map data/obj.data cfg/yolov4_custom.cfg /content/gdrive/MyDrive/YOLO/recursos/yolov4_custom_best.weights
```

Etapa 8: Testando o modelo treinado

Plota na tela a tabela contendo o identificador da imagem e algumas referências utilizadas para o tratamento das imagens, centralizando-a. Para isso:

- Carrega a tabela:

```
import pandas as pd

# Lendo e exibindo a planilia e ignorando o cabeçalho
tabela = pd.read_csv('/content/gdrive/MyDrive/ground_truth.csv')
display (tabela)
```

- Salva os dados da imagem, utilizando como referência o identificador:

```
linha = tabela.loc[tabela['img_name'] == x]
linha
```

- Salva as coordenadas do olho esquerdo e cortar a imagem:

```
img_tratada = imagem.crop((coords_eye[0], coords_eye[1], coords_eye[2], coords_eye[3]))
img_tratada
```

Execução do darknet, passando a rede neural treinada, os pesos treinados e a imagem tratada:

```
!./darknet detector test data/obj.data cfg/yolov4_custom.cfg /content/gdrive/MyDrive/YOLO/recursos/yolov4_custom_3000.weights /content/gdrive/MyDrive/imgs-20220610T145202Z-001/imgs/FRGC_04622d23.jpg
```

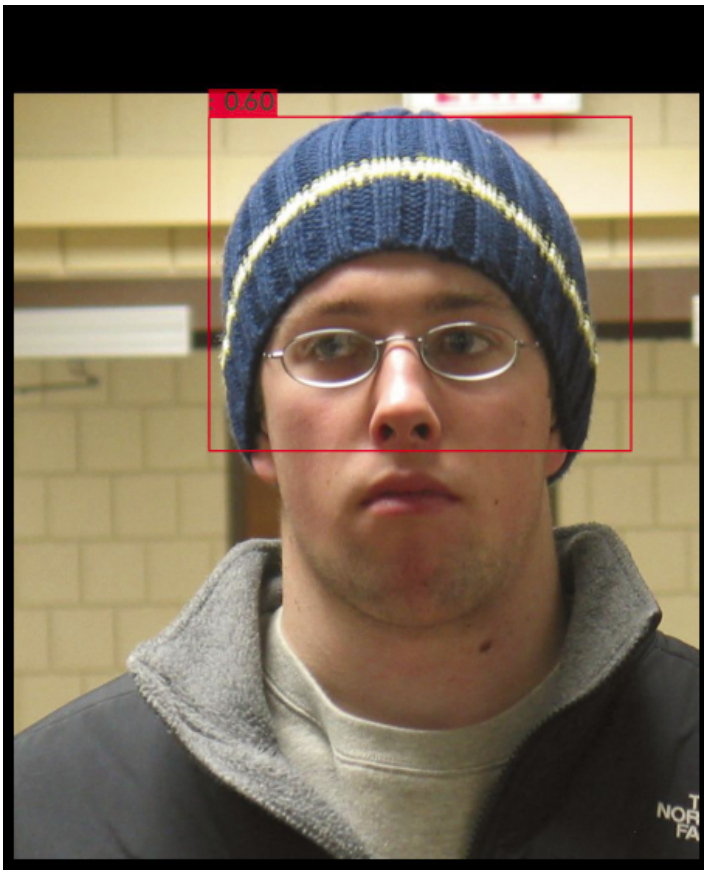
Finaliza, plotando o resultado na tela:

```
mostrar('predictions.jpg')
```

4. Resultados

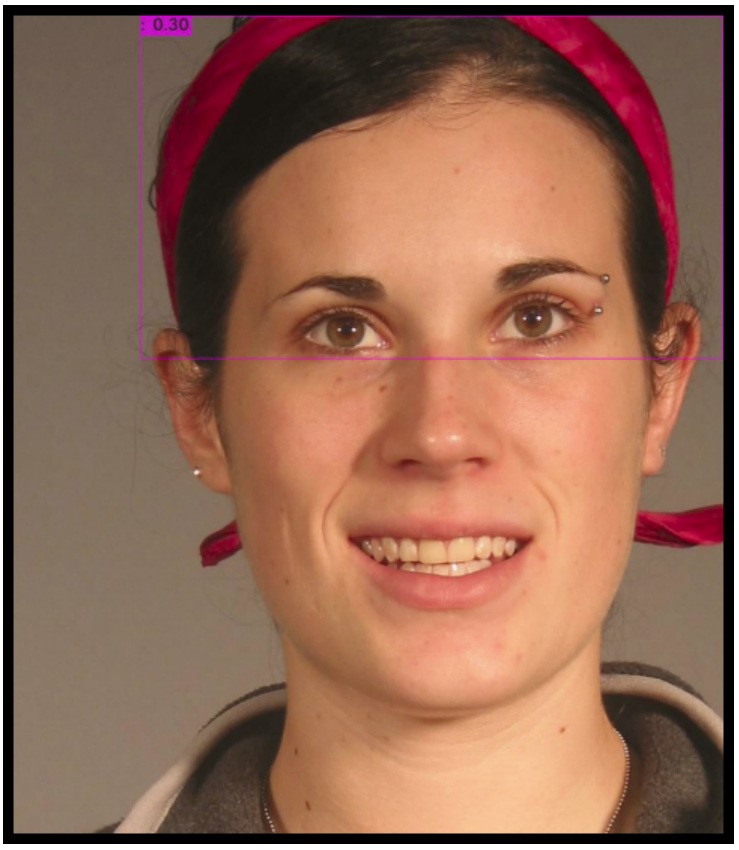
4.1. Mulher com Hijab

4.2. Homem de touca



4.3. Mulher de touca

4.4. Mulher de tiara



5. Considerações Finais

Proposto, inicialmente, o objetivo do presente trabalho é a Classificação de imagens utilizando a rede neural YOLO, a partir da framework Darknet. Ao decorrer do desenvolvimento da aplicação, foi entendido que a qualidade de processamento, em outras palavras, a precisão de acerto é diretamente direcionada pelo tempo de treinamento da rede neural. Sendo assim, para uma melhor precisão nos acertos do modelo treinado, o tempo de treinamento representou, consideravelmente, a maior etapa do projeto, logo, diante do tempo curto, não foi possível aplicar a classificação. Todavia, o desenvolvimento da classificação no presente projeto, serve de motivação para projetos futuros.

6. Referências

YOLO: Real-Time Object Detection
Disponível: **https://pjreddie.com/darknet/yolo/**

IBM, O que são redes neurais?
Disponível: **https://www.ibm.com/br-pt/cloud/learn/neural-networks**

CETAX, Data Analytics, Big Data, Data Science
Disponível: **https://www.cetax.com.br/blog/machine-learning/**

Redes Neurais Artificiais
Disponível: **https://sites.icmc.usp.br/andre/research/neural/**

Detecção de Objetos com YOLO – Uma abordagem moderna
Disponível:
https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/