

# Geared or Not Dataset:

## -Is an Employee Wearing Safety Gear?

---

### **1-Describing the data:**

- This dataset is about identifying if an employee is wearing appropriate safety gears (geared) or not before starting their work.
- The model later can be deployed on camera device (or IoT system) for real time identification of people geared or not which is very critical in industry
- This data set has 2 sub-folders. One sub-folder has pictures of people wearing a safety helmet and a vest, and the other has generic pictures of people.

-link of data from Kaggle

<https://www.kaggle.com/datasets/khananikrahman/is-an-employee-wearing-safety-gear>



Link of github project

<https://github.com/TahaFawzyElshrif/GearedOrNot>

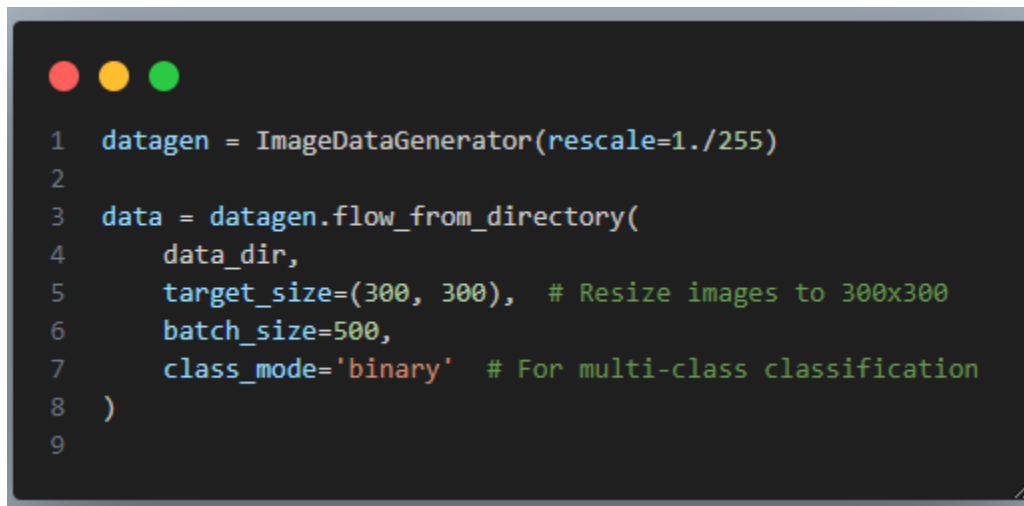
### **2-Main objective of this analysis**

- classification of image if it's geared or not (The Machine learning model implemented here)
- make model lightweight (I tried using mobile net but for this not the best results)

### **3- Analysis and DL models :**

#### **Loading and preparation :**

- Data is downloaded as folder containing geared \not geared ,so to easy load it I used ImageDataGenerator
- to make images same size I resized to 300\*300 for all images ,and rescaled by 1/255



```
1 datagen = ImageDataGenerator(rescale=1./255)
2
3 data = datagen.flow_from_directory(
4     data_dir,
5     target_size=(300, 300), # Resize images to 300x300
6     batch_size=500,
7     class_mode='binary' # For multi-class classification
8 )
9
```

- Unfortunately data is not seperated train test ,so I manually seperated it ,by getting images then used sklearn traintestsplrit (all data length is <500 so without augmentation : won't taking memory)

```

1 X, y = next(data)
2 X_train, X_, y_train, y_ = train_test_split(X, y, test_size=0.2, random_state=42)
3 X_cv, X_test, y_cv, y_test = train_test_split(X_, y_, test_size=0.5, random_state=42)
4

```

***To ensure that all loaded well : I plotted some images***

```

1 def plot_images(images_arr, shape_x, shape_y):
2     fig, axes = plt.subplots(shape_x, shape_y, figsize=(20, 20))
3     axes = axes.flatten()
4     for img, ax in zip(images_arr, axes):
5         ax.imshow(img)
6         ax.axis('off')
7     plt.tight_layout()
8     plt.show()
9

```



With this I made 2 callbacks (tensorflow implement):

1- history of accuracy ,loss

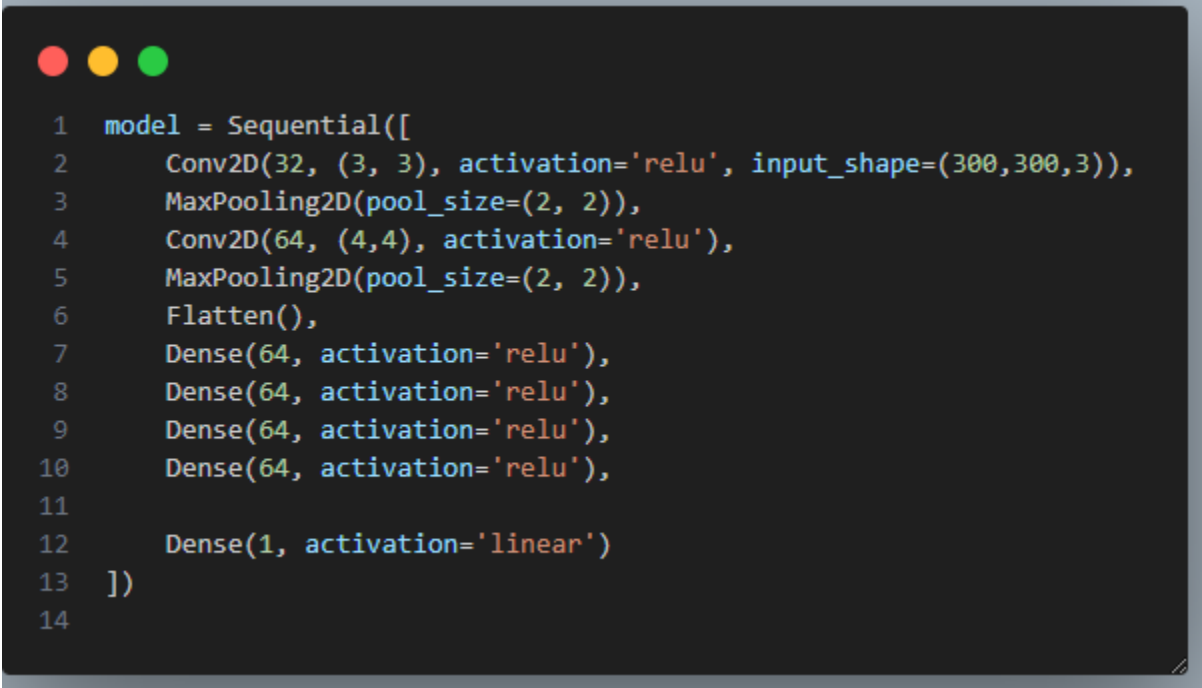
2- other to stop when getting very high accuracy (85%)

### I-Simple CNN:

The first model I have tried is simple CNN

I have tried this with two architecture :

A:

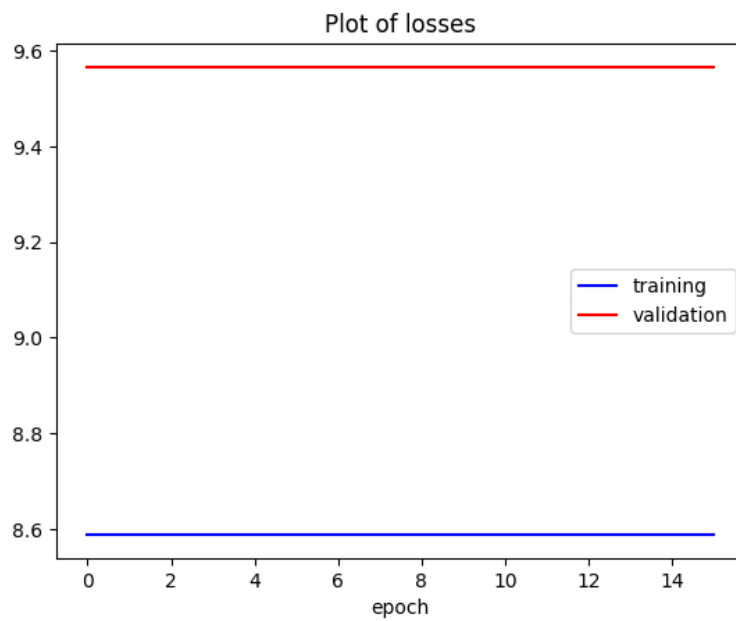
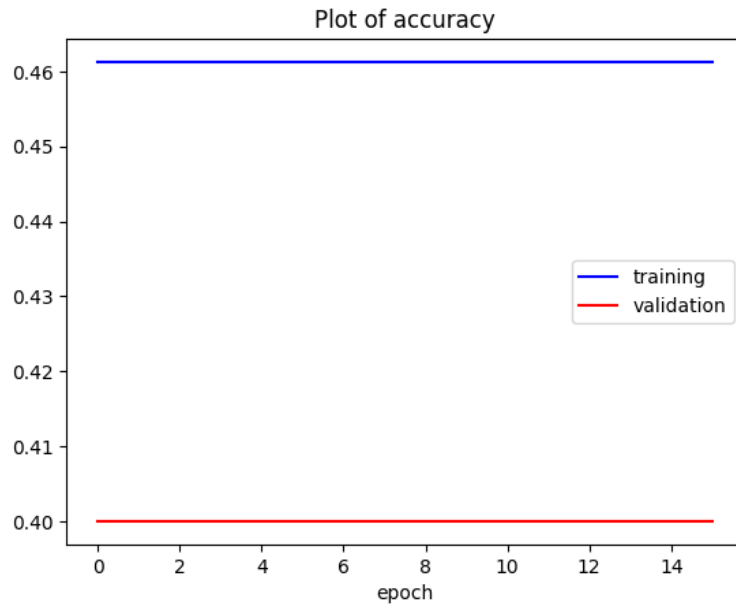


```
1  model = Sequential([
2      Conv2D(32, (3, 3), activation='relu', input_shape=(300,300,3)),
3      MaxPooling2D(pool_size=(2, 2)),
4      Conv2D(64, (4,4), activation='relu'),
5      MaxPooling2D(pool_size=(2, 2)),
6      Flatten(),
7      Dense(64, activation='relu'),
8      Dense(64, activation='relu'),
9      Dense(64, activation='relu'),
10     Dense(64, activation='relu'),
11
12     Dense(1, activation='linear')
13 ])
14
```

However the last accuracy at train , cv, test isn't very good (about 50%)

I tried to see history to check if in some point we get got accuracy ,loss so early stop

But the plot say:



Which mean very poor performance (constant) !

**B(Using pytorch):**

```

1 class SimpleCNN(nn.Module):
2     def __init__(self):
3         super(SimpleCNN, self).__init__()
4         self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=0)
5         self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=0)
6         self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=0)
7         self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
8         self.fc1 = nn.Linear(1, 512)
9         self.fc2 = nn.Linear(512, 1)
10
11
12     def forward(self, x):
13         x = self.pool((self.conv1(x)))
14         x = self.pool((self.conv2(x)))
15         x = self.pool((self.conv3(x)))
16         x = torch.flatten(x, 1)
17         self.fc1 = nn.Linear(x.size(1), 512) #
18         x = F.relu(self.fc1(x))
19         x = self.fc2(x)
20         return x
21

```

Also this model gave the same low accuracy , I also tried to play with hyper parameters ,but not getting better

## **II- VGG transfer learning**

May should I use large network with some preknowledge ?

So I tried first simple Model to transfer .

-weights :have been from imagenet data

```

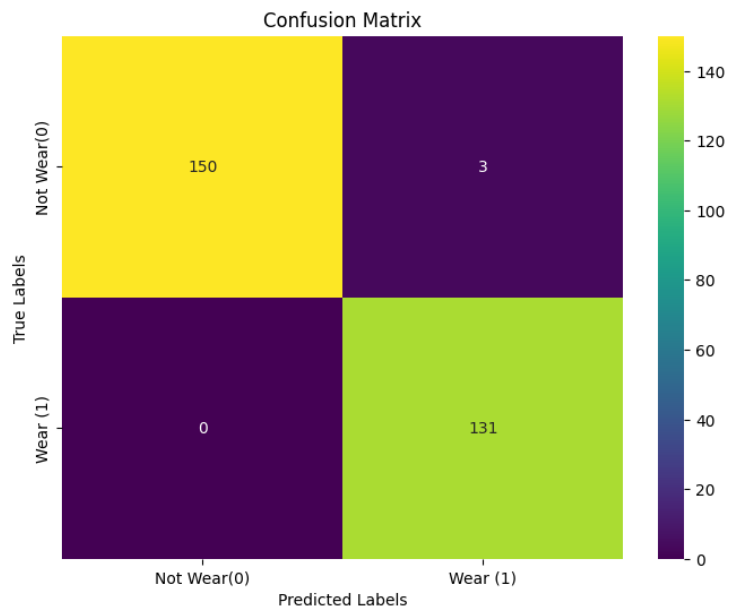
1 base_model = VGG16(weights='imagenet', include_top=False, input_shape=(300,300, 3))
2 for layer in base_model.layers:
3     layer.trainable = False
4 x = base_model.output
5 x = Flatten()(x)
6 x = Dense(512, activation='relu')(x)
7 x = Dense(256, activation='relu')(x)
8 predictions = Dense(1, activation='sigmoid')(x)
9
10 # Create the full model
11 VGG_model = Model(inputs=base_model.input, outputs=predictions)

```

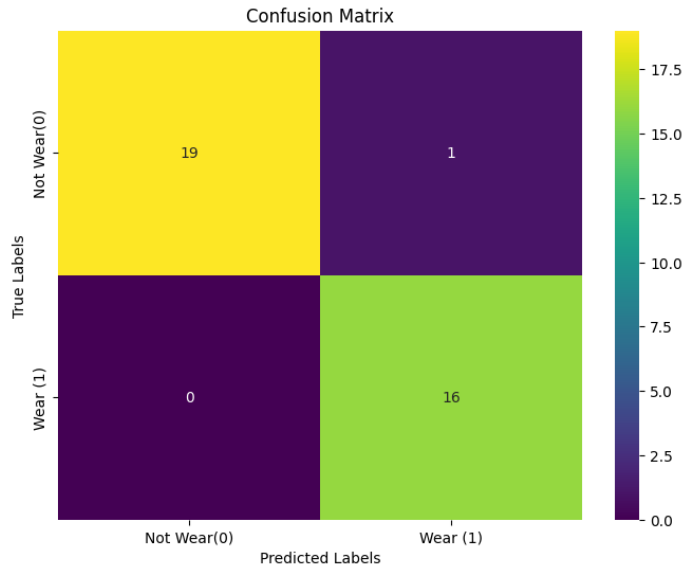
The Good thing is trained well with very large accuracy in both train ,cv,test 😊

```
Epoch 1/16
9/9 ————— 325s 36s/step - accuracy: 0.5503 - loss: 2.4634 - val_accuracy: 0.7429 - val_loss: 1.3706
Epoch 2/16
9/9 ————— 0s 31s/step - accuracy: 0.8332 - loss: 0.5540
Stopping training because accuracy exceeded 0.85
9/9 ————— 320s 36s/step - accuracy: 0.8390 - loss: 0.5371 - val_accuracy: 0.9143 - val_loss: 0.2384
2/2 ————— 35s 3s/step - accuracy: 0.9711 - loss: 0.1301
Test accuracy: 0.9722222089767456
```

Then plotted confusion matrixes in train



Test:



### III-MobileNet:

The simple VGG showed very high accuracy ,so I tried to optimize it for mobile device (Camera or microcontroller camera)

```
1 base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(300,300, 3))
2
3 # Freeze the base model layers
4 for layer in base_model.layers:
5     layer.trainable = False
6
7 x = base_model.output
8 x = Flatten()(x)
9 x = Dense(512, activation='relu')(x)
10 x = Dropout(0.7)(x)
11 x = Dense(256, activation='relu')(x)
12 x = Dropout(0.7)(x)
13 predictions = Dense(1, activation='sigmoid')(x)
14
15 # Define the model
16 Mobile_model = Model(inputs=base_model.input, outputs=predictions)
```

Also the model get very well accuracy at train ,cv data

However not that well in test 😞



```

Epoch 1/16
9/9 ██████████ 43s 4s/step - accuracy: 0.5729 - loss: 10.6682 - val_accuracy: 0.8571 - val_loss: 2.5913
Epoch 2/16
9/9 ██████████ 36s 4s/step - accuracy: 0.8197 - loss: 11.7362 - val_accuracy: 0.8571 - val_loss: 8.2188
Epoch 3/16
9/9 ██████████ 0s 5s/step - accuracy: 0.8399 - loss: 7.3961
Stopping training because accuracy exceeded 0.85
9/9 ██████████ 49s 5s/step - accuracy: 0.8432 - loss: 7.3700 - val_accuracy: 0.9143 - val_loss: 2.1230
2/2 ██████████ 41s 5s/step - accuracy: 0.5000 - loss: 0.8730
Test accuracy: 0.5

```

I tried to add dropout layer but no benefit

#### **4-Summary and key findings related to the main objective(s) of the analysis?**

-Detecting if workers wear the gears or not is very important safety thing ,so it's good to have model check it ,the best model is VGG model which is not very complicated (like ResNet) and do very well on the data

#### **5-plan of action to revisit this analysis**

- It's good to make model lightweight so we can readd layers at Mobile Net or overall finetune it ,we can instead of this use EfficientNet for target device ,may be some model or microcontroller
- the model (VGG) did very well on train ,test ,cv ,however when trying on onlie images ,it not doing very well

```

1  from tensorflow.keras.preprocessing import image
2  import numpy as np
3
4  # Load the saved model
5  model = tf.keras.models.load_model("VGG_model.h5")
6
7  def predict_uploaded_image():
8      for image_i in uploaded.keys():
9          print("_____")
10         img = image.load_img(image_i, target_size=(300, 300))
11         x = image.img_to_array(img)
12         x = np.expand_dims(x, axis=0)
13         x=x/255
14
15         predication = model.predict(x, batch_size=10)[0]
16
17
18         print(image_i+":")
19         print("Wearing" if predication > 0.5 else "Not Wearing")
20         print(f"Probability :{predication * 100}")#recalculte ???????
21
22 from google.colab import files
23 uploaded = files.upload()
24 predict_uploaded_image()

```

,you can do this by adding more data ,containing people of the same state at both wear /not wear or augmenting data and retrain models

(you may also modify probability to give better measure for both classes)