Transfer Learning in image classification:

Transfer learning

Transfer learning, is used in machine learning, it reuses a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another.We transfer the weights that a network has learned at “task A” to a new “task B.”

The general idea is to use the knowledge a model has learned from a task with a lot of available labeled training data in a new task that doesn't have much data. Instead of starting the learning process from scratch, we start with patterns learned from solving a related task.

Transfer learning is mostly used in computer vision and natural language processing tasks like sentiment analysis due to the huge amount of computational power required.

Transfer Learning process

Pre-Trained Model

We start by downloading and using an imageNet classifier; MobileNetV2 a pretrained model from TensorFlow Hub is selected.

Problem

We identify the problem based on:

- The Dataset size
- The Dataset similarity.

Fine-tune the Model

We make adjustments to further improve performance.

Pre-Trained model

Imports

We download and import appropriate repositories i.e TensorFlow Hub.

Classifier

We use a classifier model pre-trained on ImageNet benchmark dataset.

Wrap model as keras

The model downloaded is wrapped as keras layer with hub.KerasLayer.

```
!pip install tensorflow

import numpy as np
import time

import PIL.Image as Image
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

import datetime

%load_ext tensorboard
```

```
[ ] mobilenet_v2 ="https://tfhub.dev/google/tf2-preview/mobilenet_v2/classificat:
    inception_v3 = "https://tfhub.dev/google/imagenet/inception_v3/classificatio
    
    classifier_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_v3"] {ty
```

classifier_model: mobilenet_v2

How to use the pre-trained model

Train the entire model : In this case, you use the architecture of the pre-trained model and train it according to your dataset. You're learning the model from scratch, so you'll need a large dataset along with that you will also need great amount of computational power.

Train some layers and leave the others frozen : Usually, if you've a small dataset and a large number of parameters, you'll leave more layers frozen to avoid overfitting. By contrast, if the dataset is large and the number of parameters is small, you can improve your model by training more layers to the new task since overfitting is not an issue.

Freeze the convolutional base : You keep the convolutional base in its original form and then use its outputs to feed the classifier.You use the pre-trained model as a fixed feature extraction mechanism, which can be useful if you're short on computational power, your dataset is small, and/or pre-trained model solves a problem very similar to the one you want to solve.

Problem(How to train our Model)

Train the entire model if:

Large dataset,but different from the pretrained models dataset.

Train some layers if:

Large dataset and similar to the pretrained models dataset.

Wrap model as keras if:

Small dataset and different to the pretrained models dataset.

Freeze convolutional base if:

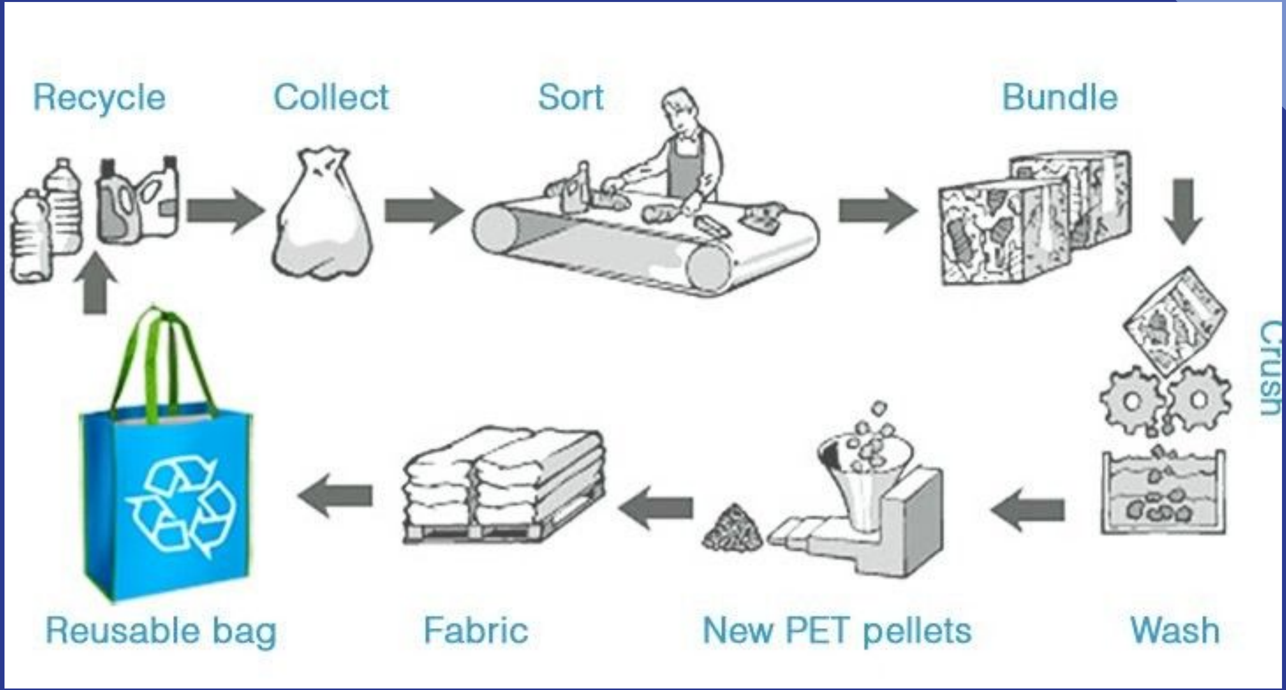
Small dataset and similar to the pretrained models dataset.

Transfer Learning for bottle recycling class recognition.

Why?

Plastic recycling made easy

The sorting of plastic during recycling is done by hand,this takes too long and requires too much labour.



Implementation

```
!pip install tensorflow

import numpy as np
import time

import PIL.Image as Image
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

import datetime

%load_ext tensorboard
```

Importing the model

Download classifier

```
mobilenet_v2 ="https://tfhub.dev/google/tf2-preview/mobilenet_v2/classificat:
inception_v3 = "https://tfhub.dev/google/imagenet/inception_v3/classificatio

classifier_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_v3"] {tyı
```

classifier_model: mobilenet_v2

```
IMAGE_SHAPE = (224, 224)

classifier = tf.keras.Sequential([
    hub.KerasLayer(classifier_model, input_shape=IMAGE_SHAPE+(3,))
])
```

```
grace_hopper = tf.keras.utils.get_file('image.jpg','https://storage.googleapis.com/download.tensorflow.org/example_images/grace_hopper.jpg')
grace_hopper = Image.open(grace_hopper).resize(IMAGE_SHAPE)
grace_hopper
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/grace_hopper.jpg
61306/61306 [==============================] - 0s 0us/step
```

Running on a single picture

```
[ ] grace_hopper = np.array(grace_hopper)/255.0
    grace_hopper.shape
```

```
(224, 224, 3)
```

```
[ ] result = classifier.predict(grace_hopper[np.newaxis, ...])
    result.shape
```

```
1/1 [==============================] - 1s 1s/step
(1, 1001)
```

```
[ ] predicted_class = tf.math.argmax(result[0], axis=-1)
    predicted_class
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=653>
```

```
[ ] labels_path = tf.keras.utils.get_file('ImageNetLabels.txt','https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt')
    imagenet_labels = np.array(open(labels_path).read().splitlines())
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt
10484/10484 [==============================] - 0s 0us/step
```

```
[ ] plt.imshow(grace_hopper)
    plt.axis('off')
    predicted_class_name = imagenet_labels[predicted_class]
    _ = plt.title("Prediction: " + predicted_class_name.title())
```

Prediction: Military Uniform

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

```
Mounted at /gdrive
```

```
[ ] import os
    DATADIR = os.path.join('/gdrive', "MyDrive")
    data_path = os.path.join(DATADIR, "datasett.zip")
    #data_path = "https://drive.google.com/file/d/1IlAazVqnYzd5ka8BTTwIhCCsjd0rG-c6/view?usp=share_link"
```

Importing the dataset from onedrive

```
[ ] import zipfile
    with zipfile.ZipFile(data_path, 'r') as zip_ref:
           zip_ref.extractall('/content/data')
```

```
[ ] batch_size = 32
    img_height = 224
    img_width = 224

    train_ds = tf.keras.utils.image_dataset_from_directory(
      str('/content/data/dataset'),
      validation_split=0.2,
      subset="training",
      seed=123,
      image_size=(img_height, img_width),
      batch_size=batch_size
    )

    val_ds = tf.keras.utils.image_dataset_from_directory(
      str('/content/data/dataset'),
      validation_split=0.2,
      subset="validation",
      seed=123,
      image_size=(img_height, img_width),
      batch_size=batch_size
    )
```

```
Found 382 files belonging to 5 classes.
Using 306 files for training.
Found 382 files belonging to 5 classes.
Using 76 files for validation.
```

Adding a dimension and passing image to model

Name	Date modified	Type	Size
disfigured	1/4/2023 5:18 PM	File folder	
good	1/4/2023 4:26 PM	File folder	
no cap	1/4/2023 7:11 PM	File folder	
no strip	1/4/2023 6:38 PM	File folder	
pigmented	1/4/2023 6:15 PM	File folder	

The data used

disfigured

good

No cap

No strip

pigmented

```
[ ] class_names = np.array(train_ds.class_names)
    print(class_names)
```

```
['disfigured' 'good' 'no cap' 'no strip' 'pigmented']
```

```
[ ] normalization_layer = tf.keras.layers.Rescaling(1./255)
    train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x-images, y-labels.
    val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x-images, y-labels.
```

```
[ ] AUTOTUNE = tf.data.AUTOTUNE
    train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
    val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

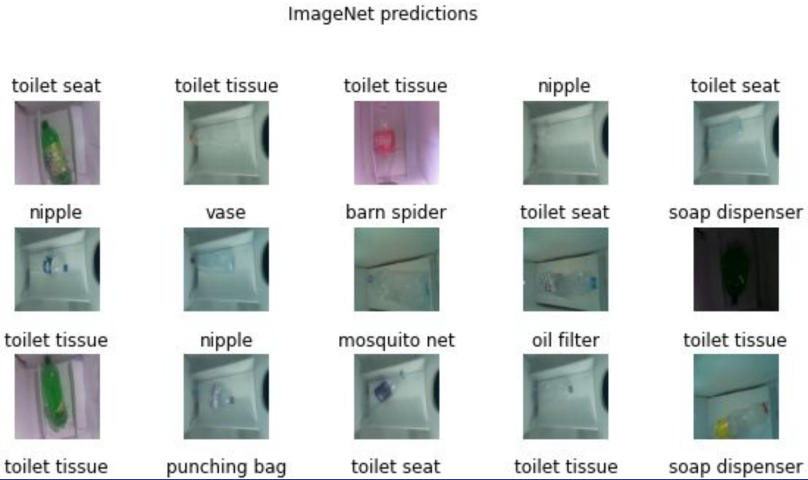
```
[ ] for image_batch, labels_batch in train_ds:
      print(image_batch.shape)
      print(labels_batch.shape)
      break
```

```
(32, 224, 224, 3)
(32,)
```

```
[ ] from keras.models import load_model

    result_batch = classifier.predict(train_ds)

    10/10 [==============================] - 8s 600ms/step

[ ] predicted_class_names = imagenet_labels[tf.math.argmax(result_batch, axis=-1)]
    predicted_class_names

        'toilet tissue', 'Band Aid', 'microwave', 'toilet tissue',
        'microwave', 'toilet seat', 'toilet tissue', 'toilet seat',
        'toilet seat', 'washer', 'modem', 'perfume', 'plastic bag',
        'nipple', 'microwave', 'toilet tissue', 'beaker', 'toilet seat',
        'toilet tissue', 'packet', 'nipple', 'nipple', 'bathtub',
        'toilet seat', 'toilet tissue', 'toilet seat', 'toilet seat',
        'toilet tissue', 'spotlight', 'hand blower', 'toilet seat',
        'cleaver', 'Band Aid', 'microwave', 'microwave', 'microwave',
        'toilet tissue', 'toilet seat', 'nipple', 'Band Aid', 'washer',
        'plastic bag', 'nipple', 'toilet seat', 'projector',
        'toilet tissue', 'toilet tissue', 'pop bottle', 'nipple',
        'toilet tissue', 'microwave', 'toilet tissue', 'shower curtain',
        'cleaver', 'mousetrap', 'washer', 'washer', 'shower curtain',
        'Band Aid', 'spotlight', 'vase', 'nipple', 'toilet seat',
        'microwave', 'toilet seat', 'toilet tissue', 'projector',
        'pop bottle', 'toilet tissue', 'shower curtain', 'toilet seat',
        'toilet seat', 'water bottle', 'toilet tissue', 'soap dispenser',
        'pedestal', 'microwave', 'toilet tissue', 'toilet tissue',
        'toilet tissue', 'bathtub', 'water bottle', 'medicine chest',
        'microwave', 'nipple', 'refrigerator', 'toilet tissue',
```

Predictions

```python
plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(image_batch[n])
  plt.title(predicted_class_names[n])
  plt.axis('off')
_ = plt.suptitle("ImageNet predictions")
```



```
mobilenet_v2 = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vec
inception_v3 = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_vec

feature_extractor_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_v3
```

feature_extractor_model: mobilenet_v2

```
feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model,
    input_shape=(224, 224, 3),
    trainable=False)
```

```
feature_batch = feature_extractor_layer(image_batch)
print(feature_batch.shape)
```

```
(32, 1280)
```

```
num_classes = len(class_names)

model = tf.keras.Sequential([
  feature_extractor_layer,
  tf.keras.layers.Dense(num_classes)
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer_1 (KerasLayer)  (None, 1280)              2257984

 dense (Dense)               (None, 5)                 6405

=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
[ ] predictions = model(image_batch)
```

```
[ ] predictions.shape
```

```
TensorShape([32, 5])
```

```
[ ] model.compile(
      optimizer=tf.keras.optimizers.Adam(),
      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      metrics=['acc'])

    log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(
        log_dir=log_dir,
        histogram_freq=1) # Enable histogram computation for every epoch.
```

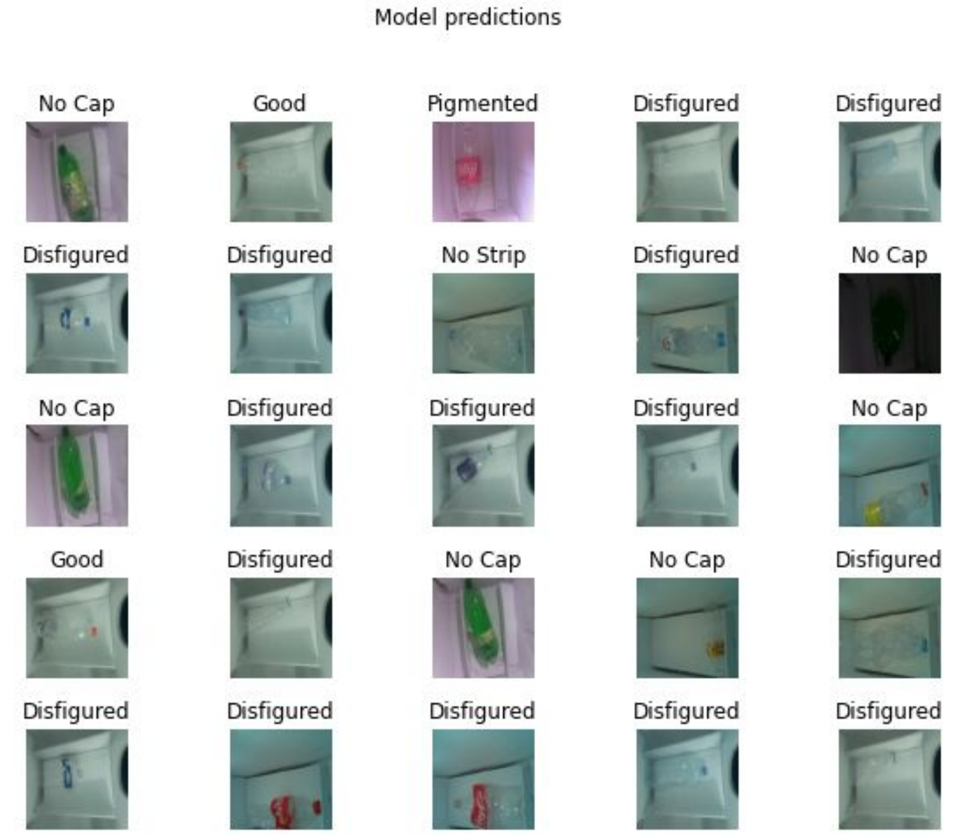
```
[ ] NUM_EPOCHS = 25

    history = model.fit(train_ds,
                        validation_data=val_ds,
                        epochs=NUM_EPOCHS,
                        callbacks=tensorboard_callback)
```

```
Epoch 17/25
10/10 [==============================] - 9s 893ms/step - loss: 0.2660 - acc: 0.8627 - val_loss: 0.8071 - val_acc: 0.6053
Epoch 18/25
10/10 [==============================] - 9s 899ms/step - loss: 0.2645 - acc: 0.8660 - val_loss: 0.8118 - val_acc: 0.5921
Epoch 19/25
10/10 [==============================] - 10s 999ms/step - loss: 0.2630 - acc: 0.8660 - val_loss: 0.8165 - val_acc: 0.5921
Epoch 20/25
10/10 [==============================] - 9s 907ms/step - loss: 0.2615 - acc: 0.8660 - val_loss: 0.8212 - val_acc: 0.5921
Epoch 21/25
10/10 [==============================] - 10s 1s/step - loss: 0.2602 - acc: 0.8660 - val_loss: 0.8259 - val_acc: 0.5921
Epoch 22/25
10/10 [==============================] - 10s 1s/step - loss: 0.2589 - acc: 0.8660 - val_loss: 0.8306 - val_acc: 0.5921
Epoch 23/25
10/10 [==============================] - 10s 1s/step - loss: 0.2576 - acc: 0.8627 - val_loss: 0.8352 - val_acc: 0.5921
Epoch 24/25
10/10 [==============================] - 10s 1s/step - loss: 0.2564 - acc: 0.8627 - val_loss: 0.8398 - val_acc: 0.5921
Epoch 25/25
10/10 [==============================] - 9s 907ms/step - loss: 0.2552 - acc: 0.8627 - val_loss: 0.8444 - val_acc: 0.5921
```

```
predicted_batch = model.predict(image_batch)
predicted_id = tf.math.argmax(predicted_batch, axis=-1)
predicted_label_batch = class_names[predicted_id]
print(predicted_label_batch)
```

```
1/1 [==============================] - 1s 632ms/step
['no cap' 'good' 'pigmented' 'disfigured' 'disfigured' 'disfigured'
 'disfigured' 'no strip' 'disfigured' 'no cap' 'no cap' 'disfigured'
 'disfigured' 'disfigured' 'no cap' 'good' 'disfigured' 'no cap' 'no cap'
 'disfigured' 'disfigured' 'disfigured' 'disfigured' 'disfigured'
 'disfigured' 'good' 'disfigured' 'no cap' 'pigmented' 'no cap'
 'disfigured' 'disfigured']
```

```
[ ] plt.figure(figsize=(10,9))
    plt.subplots_adjust(hspace=0.5)

    for n in range(30):
      plt.subplot(6,5,n+1)
      plt.imshow(image_batch[n])
      plt.title(predicted_label_batch[n].title())
      plt.axis('off')
    _ = plt.suptitle("Model predictions")
```


Model predictions
No Cap
Good
Pigmented
Disfigured
Disfigured
Disfigured
Disfigured
No Strip
Disfigured
No Cap
No Cap
Disfigured
Disfigured
Disfigured
No Cap
Good
Disfigured
No Cap
No Cap
Disfigured
Disfigured
Disfigured
Disfigured
Disfigured
Disfigured

[ ]
```
t = time.time()

export_path = "/tmp/saved_models/{}".format(int(t))
model.save(export_path)

export_path
```

```
'/tmp/saved_models/1672857323'
```

[ ]
```
reloaded = tf.keras.models.load_model(export_path)
```

[ ]
```
result_batch = model.predict(image_batch)
reloaded_result_batch = reloaded.predict(image_batch)
```

```
1/1 [==============================] - 1s 813ms/step
1/1 [==============================] - 1s 1s/step
```

[ ]
```
abs(reloaded_result_batch - result_batch).max()
```

```
2.0017843
```

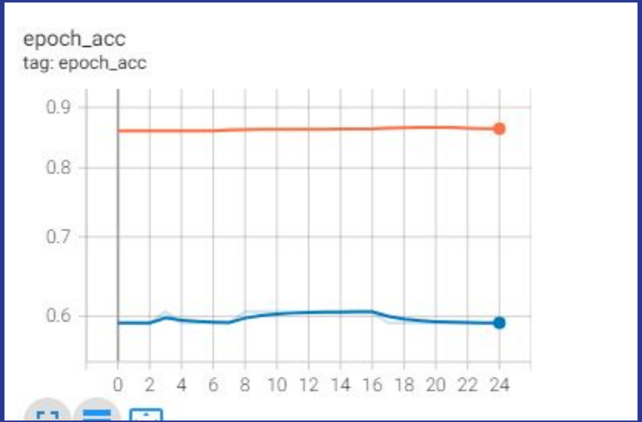
```
[ ] reloaded_predicted_id = tf.math.argmax(reloaded_result_batch, axis=-1)
    reloaded_predicted_label_batch = class_names[reloaded_predicted_id]
    print(reloaded_predicted_label_batch)
```

```
['no cap' 'good' 'pigmented' 'disfigured' 'disfigured' 'disfigured'
 'disfigured' 'no strip' 'disfigured' 'no cap' 'no cap' 'disfigured'
 'disfigured' 'disfigured' 'no cap' 'good' 'disfigured' 'no cap' 'no cap'
 'disfigured' 'disfigured' 'disfigured' 'disfigured' 'disfigured'
 'disfigured' 'good' 'disfigured' 'no cap' 'pigmented' 'no cap'
 'disfigured' 'disfigured']
```

```
[ ] plt.figure(figsize=(10,9))
    plt.subplots_adjust(hspace=0.5)
    for n in range(30):
      plt.subplot(6,5,n+1)
      plt.imshow(image_batch[n])
      plt.title(reloaded_predicted_label_batch[n].title())
      plt.axis('off')
    _ = plt.suptitle("Model predictions")
```

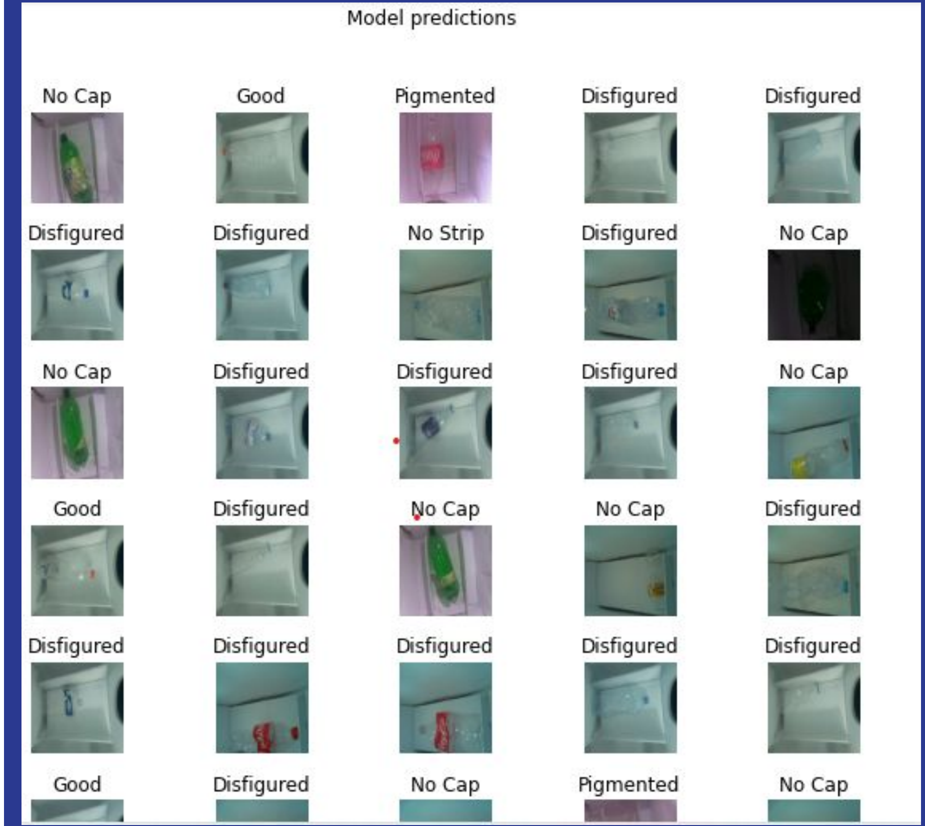

Model predictions
No Cap
Good
Pigmented
Disfigured
Disfigured
Disfigured
Disfigured
No Strip
Disfigured
No Cap
No Cap
Disfigured
Disfigured
Disfigured
No Cap
Good
Disfigured
No Cap
No Cap
Disfigured
Disfigured
Disfigured
Disfigured
Disfigured
Disfigured

Results


epoch_acc
tag: epoch_acc
0.9
0.8
0.7
0.6
0 2 4 6 8 10 12 14 16 18 20 22 24


epoch_loss
tag: epoch_loss
0.85
0.75
0.65
0.55
0.45
0.35
0.25
0 2 4 6 8 10 12 14 16 18 20 22 24

Results


Model predictions
No Cap
Good
Pigmented
Disfigured
Disfigured
Disfigured
Disfigured
No Strip
Disfigured
No Cap
No Cap
Disfigured
Disfigured
Disfigured
No Cap
Good
Disfigured
No Cap
No Cap
Disfigured
Disfigured
Disfigured
Disfigured
Disfigured
Disfigured
Good
Disfigured
No Cap
Pigmented
No Cap

Advantages of Transfer Learning

- There’s no need of large datasets for training.
- Less computational power required as compared to CNN from scratch
- Saves time.

Conclusion

Most of the predictions were accurate but each image may apply to more than one class,for example the image might not have a cap and also be disfigured but the model will only pick one class and not all the appropriate classes.So the model still has to be fine-tuned to classify the image into all the affected classes and not neglect the other properties.This might also require using a different model to classify in this manner.