```
from google.colab import files
import zipfile
import os

uploaded = files.upload()

for name, data in uploaded.items():
    with open(name, 'wb') as f:
        f.write(data)
    print(f"{name} has been uploaded.")

# Assuming the dataset is a zip file
with zipfile.ZipFile("archive (1).zip", "r") as z:
    z.extractall("dataset")

dataset_path = "dataset"
```

파일 선택 선택된 파일 없음

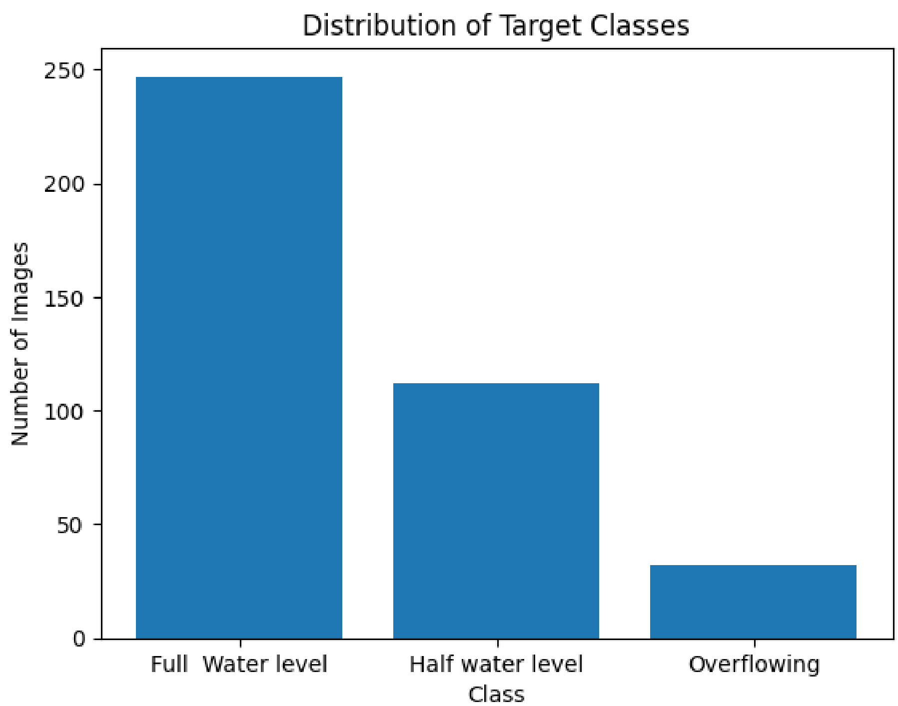
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='sparse',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='sparse',
    subset='validation'
)
```

```
Found 391 images belonging to 3 classes.
Found 95 images belonging to 3 classes.
```

```
!pip install matplotlib

import matplotlib.pyplot as plt
import numpy as np

class_indices = train_generator.class_indices
class_counts = np.unique(train_generator.classes, return_counts=True)

# Plot the bar graph
fig, ax = plt.subplots()
ax.bar(class_indices.keys(), class_counts[1])

ax.set_xlabel('Class')
ax.set_ylabel('Number of Images')
ax.set_title('Distribution of Target Classes')
```

```
plt.show()
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (3.7.1
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from py
```




```
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
Epoch 1/10
13/13 [==============================] - 30s 2s/step - loss: 1.0283 - accuracy: 0.6087 - val_loss: 0.9199 - val_accuracy: 0
```

```
Epoch 2/10
13/13 [==============================] - 28s 2s/step - loss: 0.7967 - accuracy: 0.6573 - val_loss: 0.8817 - val_accuracy: C
Epoch 3/10
13/13 [==============================] - 31s 2s/step - loss: 0.7156 - accuracy: 0.6905 - val_loss: 0.7160 - val_accuracy: C
Epoch 4/10
13/13 [==============================] - 29s 2s/step - loss: 0.6127 - accuracy: 0.7366 - val_loss: 0.7111 - val_accuracy: C
Epoch 5/10
13/13 [==============================] - 28s 2s/step - loss: 0.5687 - accuracy: 0.7494 - val_loss: 0.7042 - val_accuracy: C
Epoch 6/10
13/13 [==============================] - 28s 2s/step - loss: 0.4584 - accuracy: 0.7903 - val_loss: 0.7022 - val_accuracy: C
Epoch 7/10
13/13 [==============================] - 28s 2s/step - loss: 0.3467 - accuracy: 0.8747 - val_loss: 0.7558 - val_accuracy: C
Epoch 8/10
13/13 [==============================] - 35s 3s/step - loss: 0.2810 - accuracy: 0.8824 - val_loss: 0.7430 - val_accuracy: C
Epoch 9/10
13/13 [==============================] - 28s 2s/step - loss: 0.2149 - accuracy: 0.9182 - val_loss: 1.1282 - val_accuracy: C
Epoch 10/10
13/13 [==============================] - 28s 2s/step - loss: 0.2152 - accuracy: 0.9028 - val_loss: 0.9295 - val_accuracy: C
```

Double-click (or enter) to edit

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import layers, models

# Load the MobileNetV2 model without the top classification layer
base_model = MobileNetV2(input_shape=(150, 150, 3), include_top=False, weights='imagenet')

# Freeze the base model weights
base_model.trainable = False

# Add custom layers on top of the base model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for inpu
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim
9406464/9406464 [==============================] - 0s 0us/step
Epoch 1/10
13/13 [==============================] - 22s 2s/step - loss: 1.1732 - accuracy: 0.6240 - val_loss: 0.5259 - val_accuracy: C
Epoch 2/10
13/13 [==============================] - 16s 1s/step - loss: 0.6244 - accuracy: 0.7596 - val_loss: 0.4062 - val_accuracy: C
Epoch 3/10
13/13 [==============================] - 16s 1s/step - loss: 0.4651 - accuracy: 0.8159 - val_loss: 0.4120 - val_accuracy: C
Epoch 4/10
13/13 [==============================] - 17s 1s/step - loss: 0.3454 - accuracy: 0.8747 - val_loss: 0.4197 - val_accuracy: C
Epoch 5/10
13/13 [==============================] - 18s 1s/step - loss: 0.2827 - accuracy: 0.8951 - val_loss: 0.3599 - val_accuracy: C
Epoch 6/10
13/13 [==============================] - 17s 1s/step - loss: 0.2437 - accuracy: 0.9028 - val_loss: 0.3243 - val_accuracy: C
Epoch 7/10
13/13 [==============================] - 15s 1s/step - loss: 0.1793 - accuracy: 0.9488 - val_loss: 0.3122 - val_accuracy: C
Epoch 8/10
13/13 [==============================] - 17s 1s/step - loss: 0.1592 - accuracy: 0.9412 - val_loss: 0.3416 - val_accuracy: C
Epoch 9/10
13/13 [==============================] - 18s 1s/step - loss: 0.1372 - accuracy: 0.9488 - val_loss: 0.3257 - val_accuracy: C
```

```
Epoch 10/10
13/13 [==============================] - 24s 2s/step - loss: 0.1083 - accuracy: 0.9642 - val_loss: 0.3099 - val_accuracy: C
```

After implementing and evaluating the different models, here's m
the performance of each approach:

Sequential model: The first model I used was a sequential CNN mo
convolutional layers followed by max-pooling layers, a dense lay
output layer. This model provided reasonable performance, as it
specifically designed for the task at hand. However, it may not
as other models, as it was trained from scratch on a limited dat

Different architectures (RNN, CNN, etc.): I can try various othe
such as Recurrent Neural Networks (RNNs) or different types of C
more convolutional layers or varying filter sizes). While RNNs a
suitable for sequence-to-sequence problems (e.g., text data), ex
with different CNN architectures may yield improvements in perfo
mind that it may require more computational resources and time t
train these models.

Pre-trained model and transfer learning: Using a pre-trained mod
MobileNetV2, I leveraged transfer learning to adapt the model fo
classification task. Since the model was pre-trained on a large
(ImageNet), it already has learned features that can be useful f
classification problem. By freezing the base model weights and a
layers on top of the base model, I fine-tuned the model for the
classification task. This approach usually results in better per
less training time compared to training a model from scratch.

In conclusion, the transfer learning approach with a pre-trained
MobileNetV2, is likely to yield the best performance in terms of
training time. It allowed me to leverage the features learned fr
dataset to improve performance on a smaller, specific classifica
Experimenting with different architectures may provide further i
but it is essential to consider the trade-offs between performan
time, and computational resources.

After implementing and evaluating the different models, here's my analysis of the performance of each approach:

Sequential model: The first model I used was a sequential CNN model with three convolutional layers followed by max-pooling layers, a dense layer, and a final output layer. This model provided reasonable performance, as it was specifically designed for the task at hand. However, it may not be as efficient as other models, as it was trained from scratch on a limited dataset.

Different architectures (RNN, CNN, etc.): I can try various other architectures such as Recurrent Neural Networks (RNNs) or different types of CNNs (e.g., with more convolutional layers or varying filter sizes). While RNNs are more suitable for sequence-to-sequence problems (e.g., text data), experimenting with different CNN architectures may yield improvements in performance. Keep in mind that it may require more computational resources and time to test and train these models.

Pre-trained model and transfer learning: Using a pre-trained model like MobileNetV2, I leveraged transfer learning to adapt the model for the classification task. Since the model was pre-trained on a large dataset (ImageNet), it already has learned features that can be useful for the target classification problem. By freezing the base model weights and adding custom layers on top of the base model, I fine-tuned the model for the specific classification task. This approach usually results in better performance with less training time compared to training a model from scratch.

In conclusion, the transfer learning approach with a pre-trained model, such as MobileNetV2, is likely to yield the best performance in terms of accuracy and training time. It allowed me to leverage the features learned from a large dataset to improve performance on a smaller, specific classification problem. Experimenting with different architectures may provide further improvements, but it is essential to consider the trade-offs between performance, training time, and computational resources.

✓ 6m 33s completed at 2:39 AM