

Report Project Deep Learning:

Multiclass Image Problem



ABHRANTA PANIGRAHI · UPDATED 3 MONTHS AGO

◀ 22 ▶

New Notebook

Download (2 GB)



Urban Visual Pollution Dataset

Detect visual Pollutants like grafitti in a urban setting



Link Code: [Group 7.ipynb](#)

Anggota Kelompok

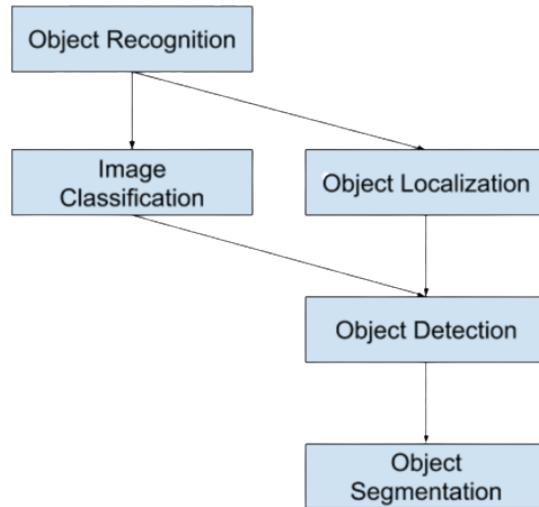
- Alyza Rahima Pramudya - 2502032125
- Shafa Amira Qonitatin - 2502009173
- Faishal Kamil - 2502001063

1) Introduction

Pencemaran lingkungan merupakan permasalahan yang kompleks dan mencakup berbagai aspek, termasuk polusi udara, air, dan tanah. Namun, dalam beberapa tahun terakhir, muncul sebuah isu baru yang semakin mendapat perhatian, yaitu polusi visual. Polusi visual merujuk pada peningkatan tampilan visual yang tidak diinginkan dalam lingkungan, seperti reklame yang berlebihan, infrastruktur yang mengganggu pandangan alam, atau perubahan drastis pada karakteristik visual suatu daerah. Akan tetapi, karena isu ini relatif baru, diperlukan penelitian yang lebih lanjut untuk mendefinisikan, memformalkan, mengukur, dan mengevaluasi polusi visual dari berbagai sudut pandang.

Atas dasar tersebut, kami melakukan dua eksperimen, **yaitu Object Recognition dan Image Classification** untuk mendeteksi dan mengklasifikasikan polusi visual. Object Recognition sendiri merupakan penggabungan antara Image Classification/ Multiclass Classification dan Object Localization. Image Classification merupakan *task* yang bertujuan untuk mengklasifikasikan gambar ke dalam suatu kategori atau kelas yang sudah ditentukan sebelumnya dan Object Localization merupakan *task* yang bertujuan untuk menemukan dan menandai letak objek dalam gambar dengan memberikan *bounding box* yang

menunjukkan area dimana objek tersebut terletak. Dengan menggabungkan Image Classification dan Object Localization, Object Recognition mampu mengenali suatu objek dalam gambar serta memberikan informasi tentang kategori objek dan lokasinya. Hal ini bisa kita lihat melalui [gambar 1](#), dimana Object Recognition mengacu pada serangkaian computer vision tasks



gambar 1.Object Recognition vs Image Classification

yang nantinya diharapkan bisa menjadi alat yang penting untuk memahami, mengatasi, dan mencari solusi untuk polusi visual. Dalam eksperimen ini, kami menggunakan [[dataset](#)] Urban Visual Pollution Dataset yang tersedia di Kaggle, dataset ini terdiri 9966 file images dan sudah terdapat train.csv yang isinya terdiri dari path untuk mengakses images di folder './images' serta class dan juga koordinat bounding box dari image path tersebut. Selain itu, terdapat juga test.csv yang isinya merupakan path untuk mengakses images yang ada di folder './images'. Berikut merupakan detail struktur dari files dan folders yang ada pada dataset

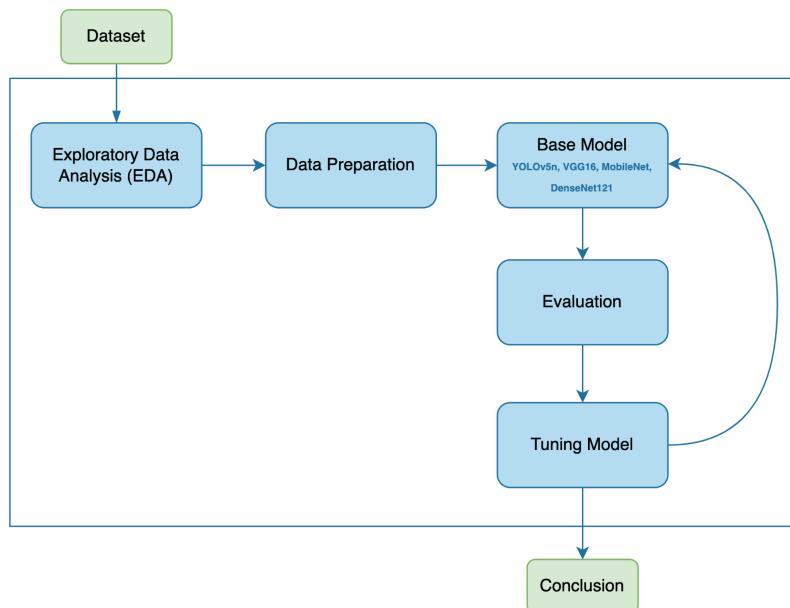
```
|--/images
|---/images
|----/*nama_file.jpg
|--/train.csv
|---/*['class', 'image_path', 'name', 'xmax', 'xmin', 'ymax', 'ymin']
|--/test.csv
|---/*['image_path']
```

/images (folders)	Folder images berisi koleksi gambar-gambar yang digunakan dalam dataset. Dimana setiap gambar memiliki nama file yang sesuai dengan nilai kolom image_path dalam file train.csv dan test.csv	
/train.csv (files)	class	Label kelas yang menunjukkan jenis polusi visual pada gambar berupa numerical
	image_path	Nama file images yang ada pada folder '/images'
	name	Nama lengkap yang menjelaskan jenis polusi visual pada gambar
	xmax	Nilai koordinat x maksimum yang menunjukkan titik paling kanan dari bounding box objek pada gambar
	xmin	Nilai koordinat x minimum yang menunjukkan titik paling kiri dari bounding box objek pada gambar
	ymax	Nilai koordinat y maksimum yang menunjukkan titik paling atas dari bounding box objek pada gambar
	ymin	Nilai koordinat y minimum yang menunjukkan titik paling bawah dari bounding box objek pada gambar
/test.csv (files)	image_path	Nama file images yang ada pada folder '/images'

Dataset ini berisi input sensor kamera mentah yang diperoleh dari sebuah armada kendaraan di wilayah geografis terbatas di KSA, yaitu Arab Saudi. Jenis-jenis polusi visual yang ada dalam dataset ini atau khususnya name dari label class yang ada pada data adalah

- | | |
|----------------------|----------------------|
| 1. GRAFFITI | 7. BAD STREETLIGHT |
| 2. FADED SIGNAGE | 8. BAD BILLBOARD |
| 3. POTHOLE | 9. SAND ON ROAD |
| 4. GARBAGE | 10. CLUTTER_SIDEWALK |
| 5. CONSTRUCTION ROAD | |

2) Metodologi

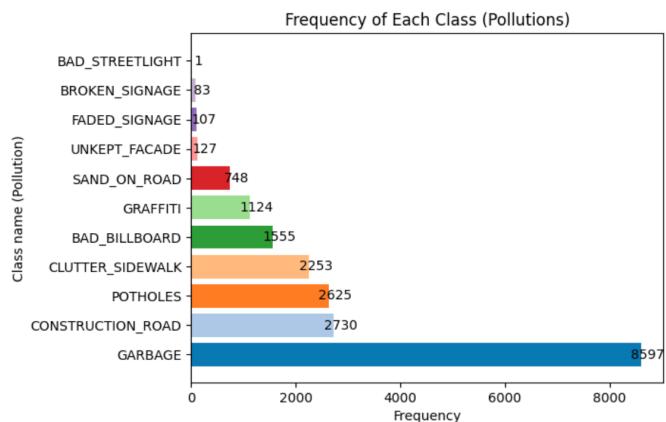


gambar 2. Alur Pengerjaan

A) Exploratory Data Analysis (EDA)

Dalam proses EDA, ditemukan bahwa jumlah images pada train set sebanyak 19950 images dan 2092 pada test set. Anomalies seperti missing value dan duplicated data tidak ditemukan di dalam kedua data tersebut, yaitu train set dan test set. Object type ditemukan ada 2 pada column di train set yaitu 'image_path' dan 'name', sisanya bentuk float dan juga ditemukan 1 object type pada satu-satunya column di test set yaitu 'image_path'.

Untuk persebaran jenis polusi visual pada train set bisa kita lihat pada gambar 3,



gambar 3. Distribusi Image untuk Tiap Kelas

Lalu, dari EDA juga ditemukan bahwa adanya bounding boxes yang tidak sesuai, contohnya terdapat bounding boxes yang melebihi ukuran images-nya dan terdapat nilai yang negatif. Untuk handle problem ini, kita akan membahas di bagian selanjutnya.



gambar 4. Bounding Box Tidak Akurat

B) Data Preparation

Pada proses Data Preparation ini, kami membuatnya menjadi ada 2 proses yaitu untuk YOLOv5n dan untuk 1. Pada dasarnya dibutuhkan perbedaan Data Preparation untuk keduanya, karena YOLOv5n termasuk Object Recognition dan VGG16, MobileNet, DenseNet121 merupakan Image Classification sehingga perbedaan-nya ada pada bounding boxes dan formatnya. Pada YOLOv5n bounding boxes masuk ke dalam features sehingga kita perlu untuk meng-preparenya. Selain itu karena format YOLOv5n berbeda sehingga kita juga perlu untuk mengubahnya dahulu menjadi .yaml.

Terlepas dari perbedaan data preparation untuk object recognition dan image classification, keduanya juga dilakukan preparation berupa pengurangan jumlah data. Seperti yang kita tahu pada awalnya dataset yang kita miliki terdiri dari 19,950 images. Namun, jumlah image ini terlalu banyak sehingga dengan resources yang terbatas, kami perlu untuk mengurangi jumlah image nya menggunakan proportional stratified

sampling hingga 5000 images. Penggunaan proportional stratified sampling digunakan agar jumlah data untuk tiap kelas tetap sesuai dengan proporsinya.

Step 1: Hitung proporsi tiap class

Garbage: $8597/5000 = 0.43$
Construction road: $2730/5000 = 0.136$
Potholes: $2625/5000 = 0.131$
Clutter sidewalk: $2253/5000 = 0.112$
Bad Billboard: $1555/5000 = 0.077$
Graffiti: $1124/5000 = 0.056$
Sand on road: $748/5000 = 0.037$
Unkept facade: $127/5000 = 0.006$
Faded signage: $107/5000 = 0.005$
Broken signage: $83/5000 = 0.004$.
Bad streetlight: $1/5000 = 0.00005$

Step 2: Hitung Jumlah Sampelnya

Garbage: $5000 * 0.43 = 2,150$
Construction road: $5000 * 0.136 = 680$
Potholes: $5000 * 0.131 = 655$
Clutter sidewalk: $5000 * 0.112 = 560$
Bad Billboard: $5000 * 0.077 = 385$
Graffiti: $5000 * 0.056 = 280$
Sand on road: $5000 * 0.037 = 185$
Unkept facade: $5000 * 0.006 = 30$
Faded signage: $5000 * 0.005 = 25$
Broken signage: $5000 * 0.004 = 20$
Bad streetlight: $5000 * 0.00005 = 0.25$

a) YOLOv5n

Seperti yang sudah dijelaskan di atas, pada data preparation yang dilakukan untuk YOLOv5n adalah dengan

1. Resized image

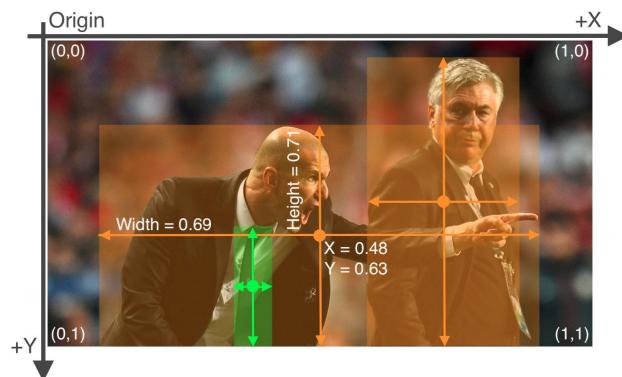
Pada YOLOv5n ini kami meng-resized images dari dataset yang kami miliki sehingga ukurannya menjadi 540 x 960. Hal ini dikarenakan gambar yang lebih besar membutuhkan

waktu yang lama untuk di-train maka dari itu dengan membuatnya lebih kecil maka akan lebih mudah untuk di-train.

Pemilihan ukuran 540 x 960 untuk YOLOv5n didasarkan karena bounding box awal terlalu kecil sehingga dibandingkan dengan mengkali koordinat xmin, xmax, ymin, dan ymax dengan 2 kami memilih untuk membagi ukuran imagennya menjadi setengahnya agar lebih cepat dilakukan.

2. Handle bounding boxes

Selanjutnya kami meng-handle bounding boxes (seperti yang sudah dijelaskan diatas). Cara kami melakukannya adalah dengan handle negative values, handle limit xvalues dan yvalues-nya dengan ukuran images yang kita miliki.



gambar 5. Bounding Box Coordinates

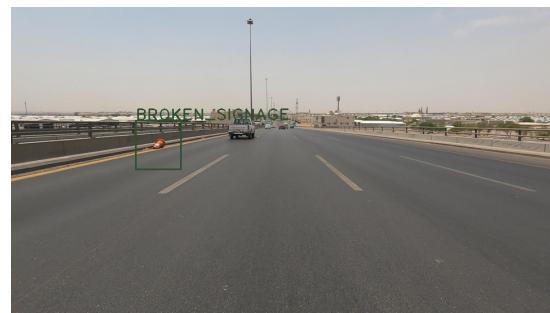
Seperti yang terlihat pada [gambar 4](#), nilai xmin, xmax, ymin, dan ymax tidak boleh negatif, sehingga apabila ditemukan nilai yang negatif akan kita ubah ke 0.

```
train_df['xmin'] = np.where(train_df['xmin'] < 0, 0,  
train_df['xmin'])  
train_df['xmax'] = np.where(train_df['xmax'] < 0, 0,  
train_df['xmax'])  
train_df['ymin'] = np.where(train_df['ymin'] < 0, 0,  
train_df['ymin'])  
train_df['ymax'] = np.where(train_df['ymax'] < 0, 0,  
train_df['ymax'])
```

Selain itu nilai xmin, xmax tidak boleh lebih besar daripada lebar gambar dan nilai ymin, ymax tidak boleh lebih besar daripada panjang gambar, oleh karena itu jika terdapat nilai xmin atau xmax yang melebihi lebar gambar akan kita ubah menjadi 960 dan apabila terdapat nilai ymin atau ymax yang melebihi tinggi gambar akan diubah menjadi 540.

```
train_df['xmin'] = np.where(train_df['xmin'] > 960 ,  
960, train_df['xmin'])  
train_df['xmax'] = np.where(train_df['xmax'] > 960,  
960, train_df['xmax'])  
train_df['ymin'] = np.where(train_df['ymin'] > 540,  
540, train_df['ymin'])  
train_df['ymax'] = np.where(train_df['ymax'] > 540,  
540, train_df['ymax'])
```

Dengan menghandle hal-hal diatas bounding box pun bisa diperbaiki hingga menjadi seperti ini, dimana bounding box bisa dengan akurat menampilkan objectnya



gambar 6. Bounding Box Akurat

3. Membuat x_center dan y_center

Dikarenakan format YOLOv5n membutuhkan x_center dan y_center pada labelnya maka kita perlu untuk membuatnya, dimana kita bisa dapatkan dengan:

```
train_df['x_center'] = (train_df['xmax'] +  
train_df['xmin']) / 2  
train_df['y_center'] = (train_df['ymax'] +  
train_df['ymin']) / 2
```

4. Membuat height dan width

Dikarenakan format YOLOv5n membutuhkan height dan width pada labelnya maka kita perlu untuk membuatnya, dimana kita bisa dapatkan dengan:

```
train_df['width'] = train_df['xmax'] -  
train_df['xmin']  
train_df['height'] = train_df['ymax'] -  
train_df['ymin']
```

Selain itu, kami juga melakukan normalisasi agar rentang nilai untuk labelnya berada rentang 0 dan 1 pada data train dan validation karena diperlukan untuk format YOLO dengan:

```
train_df['x_center'] = train_df.apply(lambda row:  
((row['xmax'] + row['xmin']) / (2 * 960)), axis=1)  
train_df['y_center'] = train_df.apply(lambda row:  
((row['ymax'] + row['ymin']) / (2 * 540)), axis=1)  
train_df['width'] = train_df.apply(lambda row:  
((row['xmax'] - row['xmin']) / 960), axis=1)  
train_df['height'] = train_df.apply(lambda row:  
((row['ymax'] - row['ymin']) / 540), axis=1)
```

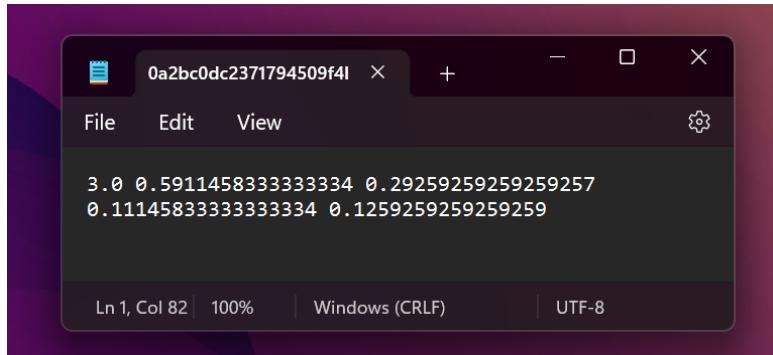
5. Split train.csv menjadi train set dan validation set

Karena kami berpikir akan menggunakan train set dan validation set nya dengan berulang-ulang dan akan menyimpannya di sebuah folder baru, maka kami splitnya dengan tidak random, yaitu dengan menghitung 80% dari dataset train akan menjadi train set dan sisanya, yaitu 20% akan menjadi validation set, dengan:

```
print("80% from 4970:", int(4970*0.8))
```

```
train = trainReduced_df[0:3976]
valid = trainReduced_df[3976:]
```

Dengan data preparation yang sudah kami lakukan di atas, selanjutnya kami perlu untuk meng-convert image class, x_center, y_center, width, dan heightnya menjadi 1 file txt untuk 1 buah image.



gambar 7. YOLO Format

Selain harus membuat 1 file txt untuk 1 buah image, kita juga perlu membuat file .yaml jika kita ingin menggunakan YOLOv5. file .yaml ini akan digunakan oleh model untuk mengetahui dimana letak image pada train, val, dan test set. Selain itu file ini juga memberitahukan jumlah classnya dan nama untuk setiap classnya seperti ini.

```
train: /content/drive/MyDrive/DeepLearningProject/PROJECT/model/train/images
val: /content/drive/MyDrive/DeepLearningProject/PROJECT/model/valid/images
test: /content/drive/MyDrive/DeepLearningProject/PROJECT/model/test/images

nc: 11,
names: ['GARBAGE', 'CONSTRUCTION_ROAD', 'POTHOLE', 'CLUTTER_SIDEWALK',
'BAD_BILLBOARD', 'GRAFFITI', 'SAND_ON_ROAD', 'UNKEPT_FAÇADE', 'FADED_SIGNAGE',
'BROKEN_SIGNAGE', 'BAD_STREETLIGHT']
```

b) VGG16, MobileNet, DenseNet121

Data preparation yang dilakukan untuk pembuatan model-model ini melibatkan tiga langkah. Pertama, dilakukan proses resize untuk mengubah ukuran gambar menjadi 640x640. Selanjutnya, data yang sudah di-resize dibagi menjadi tiga bagian: train set (80%), valid set (10%), dan test set (10%), dengan proporsi yang telah disebutkan. Terakhir,

dibuatlah generator khusus untuk train set, valid set, dan test set, yang akan digunakan dalam proses pelatihan dan evaluasi model.

Ukuran Image perlu di resize mengingat kita memiliki sumber daya yang terbatas, apabila image tidak di resize maka runtime akan terputus karena melebihi RAM yang disediakan oleh platform. Selanjutnya kami membuat generator untuk train set, valid set, dan test set menggunakan code berikut:

```
train_datagen = ImageDataGenerator(rescale=1.0/255)
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    directory=image_folder,
    x_col='image_path',
    y_col='name',
    class_mode='categorical',
    target_size=image_size,
    batch_size=batch_size
)
valid_datagen = ImageDataGenerator(rescale=1.0/255)
valid_generator = valid_datagen.flow_from_dataframe(
    dataframe=val_df,
    directory=image_folder,
    x_col='image_path',
    y_col='name',
    class_mode='categorical',
    target_size=image_size,
    batch_size=batch_size
)
test_datagen = ImageDataGenerator(rescale=1.0/255)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=test_df,
    directory=image_folder,
    x_col='image_path',
    y_col='name',
    class_mode='categorical',
    target_size=image_size,
    batch_size=batch_size
)
```

Untuk membuat generator, langkah-langkahnya adalah sebagai berikut: Pertama, kita menyusun daftar path (lokasi) gambar dan label yang sesuai untuk setiap set data. Setiap set data (train, valid, dan test) kemudian diinisialisasi dengan ImageDataGenerator yang akan melakukan penyesuaian skala pada nilai piksel gambar. Generator train, valid, dan test kemudian dibuat menggunakan fungsi flow_from_dataframe dengan memberikan dataframe, path folder, kolom gambar, kolom label, mode kelas, ukuran target gambar, dan ukuran batch sebagai parameter.

C) Base Model

a) YOLOv5n

Pada YOLOv5n ini, step yang perlu dilakukan adalah dengan meng- clone github yolov5 dan install requirements.txt nya. Selanjutnya kita bisa langsung men-training modelnya dengan menggunakan code berikut:

```
!python train.py --batch 8 --epochs 5 --data ./data.yaml
--weights yolov5n.pt --img 960
```

```

      from   n    params   module           arguments
0        -1   1     1760  models.common.Conv  [3, 16, 6, 2, 2]
1        -1   1     4672  models.common.Conv  [16, 32, 3, 2]
2        -1   1     4800  models.common.C3   [32, 32, 1]
3        -1   1    18560  models.common.Conv  [32, 64, 3, 2]
4        -1   2    29184  models.common.C3   [64, 64, 2]
5        -1   1    73984  models.common.Conv  [64, 128, 3, 2]
6        -1   3   156928  models.common.C3   [128, 128, 3]
7        -1   1    295424  models.common.Conv  [128, 256, 3, 2]
8        -1   1    296448  models.common.C3   [256, 256, 1]
9        -1   1   164608  models.common.SPPF  [256, 256, 5]
10       -1   1    33024  models.common.Conv  [256, 128, 1, 1]
11       -1   1      0  torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
12      [-1, 6]  1      0  models.common.Concat [1]
13      -1   1    90880  models.common.C3   [256, 128, 1, False]
14      -1   1     8320  models.common.Conv  [128, 64, 1, 1]
15      -1   1      0  torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
16      [-1, 4]  1      0  models.common.Concat [1]
17      -1   1    22912  models.common.C3   [128, 64, 1, False]
18      -1   1    36992  models.common.Conv  [64, 64, 3, 2]
19      [-1, 14] 1      0  models.common.Concat [1]
20      -1   1    74496  models.common.C3   [128, 128, 1, False]
21      -1   1   147712  models.common.Conv  [128, 128, 3, 2]
22      [-1, 10] 1      0  models.common.Concat [1]
23      -1   1    296448  models.common.C3   [256, 256, 1, False]
24      [17, 20, 23] 1   21648  models.yolo.Detect [11, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [
```

Model summary: 214 layers, 1778800 parameters, 1778800 gradients, 4.3 GFLOPs

```

Epoch    GPU_mem   box_loss   obj_loss   cls_loss   Instances   Size
  0/4      OG       0.09758   0.02728   0.04785      10          960: 100% 363/363 [57:44<00:00,  9.54s/it]
           Class     Images Instances      P          R          mAP50   mAP50-95: 100% 50/50 [03:17<00:00,  3.95s/it]
           all        793      793      0.000159      0.141        0.00172  0.000364

Epoch    GPU_mem   box_loss   obj_loss   cls_loss   Instances   Size
  1/4      OG       0.08009   0.02294   0.0209      8          960: 100% 363/363 [57:10<00:00,  9.45s/it]
           Class     Images Instances      P          R          mAP50   mAP50-95: 100% 50/50 [03:10<00:00,  3.81s/it]
           all        793      793      0.000334      0.0783        0.000322  7.9e-05

Epoch    GPU_mem   box_loss   obj_loss   cls_loss   Instances   Size
  2/4      OG       0.07695   0.02215   0.01388     12          960: 100% 363/363 [59:18<00:00,  9.80s/it]
           Class     Images Instances      P          R          mAP50   mAP50-95: 100% 50/50 [03:12<00:00,  3.85s/it]
           all        793      793      0.0338      0.0101        0.00195  0.000448

Epoch    GPU_mem   box_loss   obj_loss   cls_loss   Instances   Size
  3/4      OG       0.07364   0.02167   0.01164     12          960: 100% 363/363 [59:39<00:00,  9.86s/it]
           Class     Images Instances      P          R          mAP50   mAP50-95: 100% 50/50 [03:10<00:00,  3.80s/it]
           all        793      793      0.341      0.0152        0.00175  0.000312

Epoch    GPU_mem   box_loss   obj_loss   cls_loss   Instances   Size
  4/4      OG       0.07164   0.02181   0.01012      8          960: 100% 363/363 [58:18<00:00,  9.64s/it]
           Class     Images Instances      P          R          mAP50   mAP50-95: 100% 50/50 [03:06<00:00,  3.74s/it]
           all        793      793      0.000275      0.0581        0.000293  6.12e-05

5 epochs completed in 5.137 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 4.0MB
Optimizer stripped from runs/train/exp/weights/best.pt, 4.0MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 1774048 parameters, 0 gradients, 4.2 GFLOPs
      Class     Images Instances      P          R          mAP50   mAP50-95: 100% 50/50 [02:56<00:00,  3.53s/it]
      all        793      793      0.338      0.0101        0.00195  0.00045
      GARBAGE    793      233      0          0          0          0
      CONSTRUCTION_ROAD 793      2         0          0          0          0
      UNKEPT_FAÇADE   793      340      1         0          0          0
      FADED_SIGNAGE   793      146      1         0          0          0
      BROKEN_SIGNAGE   793      66       0.0272      0.0606        0.0117  0.0027
      BAD_STREETLIGHT 793      6         0          0          0          0

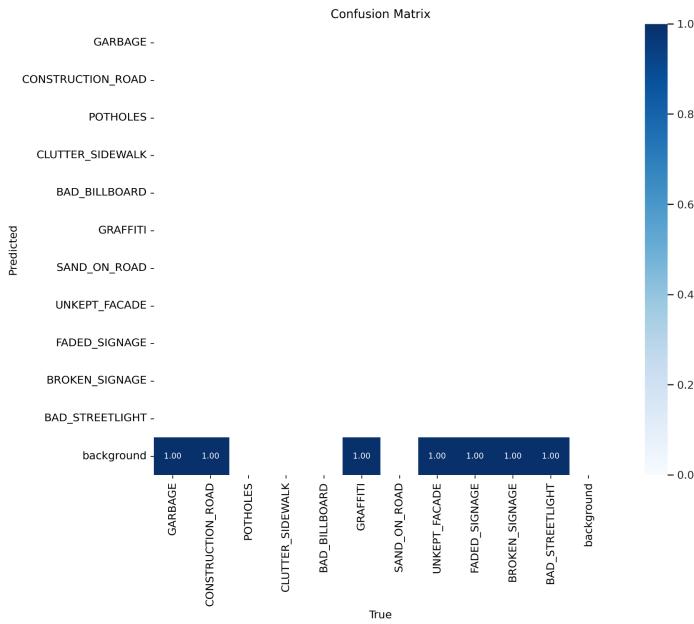
Results saved to runs/train/exp

```

gambar 8. Hasil Train Menggunakan YOLOV5n

Pertimbangan untuk mengambil epochnya 5 adalah karena kami melihat progress epochnya sangat lama. Maka dari itu, kami berencana untuk memulainya dari epoch 5 saja dahulu dan ternyata untuk 5 epoch membutuhkan 5 jam untuk training modelnya dan hasil yang didapatkan juga sangat jelek, diindikasikan dengan nilai presisi sebesar 0,338, recall 0,0101, mAP50 0,00195, dan mAP50-95 0,00045.

Selain itu kinerja buruk model dapat dilihat dari confusion matrix dan perbandingan antara ground truth dengan hasil prediksinya. Seperti yang bisa kita lihat pada gambar confusion matrix di bawah, model memprediksi gambar hanya sebagai background dan tidak bisa menemukan object yang seharusnya ia prediksi.



gambar 9. Corelation Matrix

Begini pula pada gambar perbandingan antara ground truth dan hasil prediksinya. Kita bisa melihat bahwa model kita tidak bisa memprediksi object yang seharusnya ada pada gambar tersebut.



gambar 10. Ground Truth vs Prediction

Kami menduga penyebab dari hasil yang didapatkan sangat buruk adalah karena epoch-nya yang sangat kecil, yaitu 5. Namun, dikarenakan waktu train yang sangat lama dan keterbatasan RAM, kami berpikir untuk tidak melanjutkan eksperimen ini menggunakan Object Recognition dan melanjutkan eksperimen menjadi Multiclass Image Classification.

b) VGG16

Selanjutnya kami membuat model dengan VGG16, kami melakukan epoch sebanyak 10 kali dengan menggunakan pretrained weights dari dataset imageNet.

Pada model ini kami membuka/mematikan training pada semua layer pretrained dan menambah layer-layer kustom untuk melakukan klasifikasi pada data. Pertama kami mengambil output dari modelBuilder (VGG16(weights='imagenet', include_top=False, input_shape=(640, 640, 3))) sebagai input pada layer berikutnya. Kemudian, dilakukan global average pooling untuk mengurangi dimensi dari output menjadi vektor fitur tunggal. Selanjutnya, ditambahkan fully connected layer dengan 256 unit dan fungsi aktivasi ReLU dan yang terakhir kami tambahkan fully connected layer (Dense(num_classes, activation='softmax')) dengan num_classes unit (10) dan fungsi aktivasi softmax untuk menghasilkan probabilitas image class.

Hasil akurasi atau keakuratan dari model ini adalah 0.6222 atau 62.22% pada train set dan 0.5887 atau 58.87% pada validation set. Dikarenakan hasil akurasinya masih kurang baik kami akan melakukan tuning pada model dengan menambahkan jumlah epochnya. Terkait parameter, model VGG16 ini memiliki jumlah parameter sebanyak 14.848.586

```
=====
Total params: 14,848,586
Trainable params: 133,898
Non-trainable params: 14,714,688
```

gambar 11. Jumlah Parameter VGG16

c) MobileNet

Selanjutnya kami membuat model dengan mobileNet, kami melakukan epoch sebanyak 10 kali dengan menggunakan pretrained weights dari dataset imageNet.

Pada model ini kami membekukan/mematikan training pada semua layer pretrained dan menambah layer-layer kustom untuk melakukan klasifikasi pada data. Pertama kami mengambil output dari `baseModel (MobileNet(weights='imagenet', include_top=False, input_shape=(640, 640, 3)))` sebagai input pada layer berikutnya. Kemudian, dilakukan global average pooling untuk mengurangi dimensi dari output menjadi vektor fitur tunggal. Selanjutnya, ditambahkan fully connected layer dengan 256 unit dan fungsi aktivasi ReLU dan yang terakhir kami tambahkan fully connected layer (`Dense(num_classes, activation='softmax')`) dengan `num_classes` unit (10) dan fungsi aktivasi softmax untuk menghasilkan probabilitas image class.

Hasil akurasi atau keakuratan dari model ini adalah 0.7487 atau 74.87% pada train set dan 0.6734 atau 67.34% pada validation set. Dikarenakan hasil akurasinya masih kurang baik kami akan melakukan tuning pada model dengan menambahkan jumlah epochnya. Terkait parameter, model mobileNet ini memiliki jumlah parameter sebanyak 3.493.834

```
=====
Total params: 3,493,834
Trainable params: 264,970
Non-trainable params: 3,228,864
```

gambar 12. Jumlah Parameter mobileNet

d) DenseNet121

Selanjutnya kami membuat model dengan DenseNet121, kami melakukan epoch sebanyak 10 kali dengan menggunakan pretrained weights dari dataset imageNet.

Pada model ini kami membekukan/mematikan training pada semua layer pretrained dan menambah layer-layer kustom untuk melakukan klasifikasi pada data. Pertama kami mengambil output dari `baseModel` yang didapat dari kode berikut (`DenseNet121(weights='imagenet',include_top=False,input_shape=(640, 640, 3))`) sebagai input pada layer berikutnya. Kemudian,

dilakukan global average pooling untuk mengurangi dimensi dari output menjadi vektor fitur tunggal. Selanjutnya, ditambahkan fully connected layer dengan 256 unit dan fungsi aktivasi ReLU dan yang terakhir kami tambahkan fully connected layer (Dense(num_classes, activation='softmax')) dengan num_classes unit (10) dan fungsi aktivasi softmax untuk menghasilkan probabilitas image class.

Hasil akurasi atau keakuratan dari model ini adalah 0.6965 atau 69.65% pada train set dan 0.6431 atau 64.31% pada validation set. Dikarenakan hasil akurasinya masih kurang baik kami akan melakukan tuning pada model dengan menambahkan jumlah epochnya. Terkait parameter, model denseNet121 ini memiliki jumlah parameter sebanyak 7.302.474

```
=====
Total params: 7,302,474
Trainable params: 264,970
Non-trainable params: 7,037,504
```

gambar 13. Jumlah Parameter denseNet121

D) Tuning Model

a) VGG16

Dalam melakukan tuning model, kami akan membuat model VGG16 baru dengan cara yang sama seperti sebelumnya, yaitu dengan cara membekukan/mematikan training pada semua layer pretrained dan menambah layer-layer kustom untuk melakukan klasifikasi pada data. Pertama kami mengambil output dari modelBuilder (VGG16(weights='imagenet', include_top=False, input_shape=(640, 640, 3))) sebagai input pada layer berikutnya. Kemudian, dilakukan global average pooling untuk mengurangi dimensi dari output menjadi vektor fitur tunggal. Selanjutnya, ditambahkan fully connected layer dengan 256 unit dan fungsi aktivasi ReLU dan yang terakhir kami tambahkan fully connected layer (Dense(num_classes, activation='softmax')) dengan

`num_classes` unit (10) dan fungsi aktivasi softmax untuk menghasilkan probabilitas image class.

Hal yang membedakannya dengan base model adalah dari jumlah epochnya. Jika pada base model kami menggunakan epoch sebanyak 10 kali, ketika di tuning kami menggunakan epoch sebanyak 30 kali. Hasil tuning ini menunjukkan dampak baik dilihat dari nilai akurasinya yang bertambah dari yang semula.

Hasil akurasi dari model tuning ini adalah 0.6826 atau 68.26% pada train set dan 0.6310 atau 63.10% pada validation set.

b) MobileNet

Dalam melakukan tuning model, kami akan membuat model mobileNet baru dengan cara yang sama seperti sebelumnya, yaitu dengan cara membuka/mematikan training pada semua layer pretrained dan menambah layer-layer kustom untuk melakukan klasifikasi pada data. Pertama kami mengambil output dari `baseModel (MobileNet(weights='imagenet', include_top=False, input_shape=(640, 640, 3)))` sebagai input pada layer berikutnya. Kemudian, dilakukan global average pooling untuk mengurangi dimensi dari output menjadi vektor fitur tunggal. Selanjutnya, ditambahkan fully connected layer dengan 256 unit dan fungsi aktivasi ReLU dan yang terakhir kami tambahkan fully connected layer (`Dense(num_classes, activation='softmax')`) dengan `num_classes` unit (10) dan fungsi aktivasi softmax untuk menghasilkan probabilitas image class.

Hal yang membedakannya dengan base model adalah dari jumlah epochnya. Jika pada base model kami menggunakan epoch sebanyak 10 kali, ketika di tuning kami menggunakan epoch sebanyak 30 kali. Hasil tuning ini menunjukkan dampak baik dilihat dari nilai akurasinya yang bertambah dari yang semula.

Hasil akurasi dari model tuning ini adalah 0.8765 atau 87.65% pada train set dan 0.6552 atau 65.52% pada validation set.

c) DenseNet121

Dalam melakukan tuning model, kami akan membuat model mobileNet baru dengan cara yang sama seperti sebelumnya, yaitu dengan cara membuka/mematikan training pada semua layer pretrained dan menambah layer-layer kustom untuk melakukan klasifikasi pada data. Pertama kami mengambil output dari `baseModel(DenseNet121(weights='imagenet',include_top=False,input_shape=(640, 640, 3)))` sebagai input pada layer berikutnya. Kemudian, dilakukan global average pooling untuk mengurangi dimensi dari output menjadi vektor fitur tunggal. Selanjutnya, ditambahkan fully connected layer dengan 256 unit dan fungsi aktivasi ReLU dan yang terakhir kami tambahkan fully connected layer (`Dense(num_classes, activation='softmax')`) dengan `num_classes` unit (10) dan fungsi aktivasi softmax untuk menghasilkan probabilitas image class.

Hal yang membedakannya dengan base model adalah dari jumlah epochnya. Jika pada base model kami menggunakan epoch sebanyak 10 kali, ketika di tuning kami menggunakan epoch sebanyak 30 kali. Hasil tuning ini menunjukkan dampak baik dilihat dari nilai akurasinya yang bertambah dari yang semula.

Hasil akurasi dari model tuning ini adalah 0.8182 atau 81.82% pada train set dan 0.6431 atau 64.31% pada validation set.

E) Evaluation

Pada tahap evaluasi ini, kami akan menggunakan model-model yang telah dibuat sebelumnya untuk mengklasifikasikan gambar-gambar dari test set. Kemudian, kami akan melihat seberapa baik performa model dengan mengukur skor akurasi, presisi, recall, dan F1-score. Terakhir, kami akan menghitung berapa banyak model dapat memprediksi kelas semua image pada train set dengan benar dan berapa banyak yang salah di klasifikasikan.

a) VGG16

Pada Base Model dengan menggunakan VGG16 dan jumlah epoch sebanyak 10 kali dapat kita lihat bahwa nilai accuracy, precision, recall, dan f1 scorenya sangat rendah hasil evaluation-nya menunjukkan bahwa model kita hanya memiliki tingkat akurasi sebesar 24.40%, menunjukkan kinerja yang rendah dalam mengklasifikasikan data. Baik presisi maupun recall juga memiliki nilai yang rendah, dengan f1-score sebesar 9.23%. Model ini memiliki hasil klasifikasi yang seragam di seluruh kelas dan terdapat 291 prediksi yang benar serta 205 prediksi yang salah, namun performanya masih kurang baik.

```
Accuracy: 0.2439516129032258
Precision: 0.09062774791236122
Recall: 0.09658616291434992
F1-score: 0.09230830352897194
```

	precision	recall	f1-score	support
0	0.06	0.10	0.08	41
1	0.00	0.00	0.00	1
2	0.17	0.14	0.15	59
3	0.12	0.11	0.12	70
4	0.00	0.00	0.00	1
5	0.49	0.48	0.49	229
6	0.00	0.00	0.00	19
7	0.10	0.16	0.12	58
8	0.00	0.00	0.00	18
accuracy			0.28	496
macro avg	0.10	0.11	0.11	496
weighted avg	0.28	0.28	0.28	496

```
Correct predictions: 291
Wrong predictions: 205
```

gambar 14. Evaluasi VGG16 Epoch 10 Pada Test Set

Setelah melakukan tuning pada model dengan menggunakan model yang sama, yaitu VGG16 dan penambahan jumlah epoch dari 10 epoch menjadi 30 epoch, terdapat peningkatan pada akurasi menjadi 30.44%. Namun, presisi dan recall masih rendah, dengan f1-score sebesar 10.06%. Hasil klasifikasi model ini masih memiliki nilai yang seragam di seluruh kelas dan terdapat 323 prediksi yang benar serta 173 prediksi yang salah,

tetapi belum mencapai tingkat kinerja yang memadai. Oleh karena itu, kami masih perlu melakukan pengembangan lebih lanjut untuk meningkatkan performa model ini.

```
Accuracy: 0.30443548387096775
Precision: 0.09949983635041475
Recall: 0.10708589869458013
F1-score: 0.10058551822746437
```

```
y_pred = model.predict_generator(valid2_generator)
precision    recall   f1-score   support
0            0.13     0.15     0.14      41
1            0.00     0.00     0.00       1
2            0.10     0.05     0.07      59
3            0.13     0.10     0.11      70
4            0.00     0.00     0.00       1
5            0.48     0.57     0.52     229
6            0.00     0.00     0.00      19
7            0.11     0.17     0.14      58
8            0.00     0.00     0.00      18

accuracy                           0.32      496
macro avg                           0.11      496
weighted avg                          0.28      496

Correct predictions: 323
Wrong predictions: 173
```

gambar 15. Evaluasi VGG16 Epoch 30 Pada Test Set

b) MobileNet

Pada Base Model dengan menggunakan MobileNet dan jumlah epoch sebanyak 10 kali dapat kita lihat bahwa nilai accuracy, precision, recall, dan f1 scorenya sangat rendah hasil evaluation-nya menunjukkan bahwa model kita hanya memiliki tingkat akurasi sebesar 27.62%. Meskipun terdapat peningkatan dalam jumlah prediksi yang benar menjadi 335 dan prediksi yang salah menjadi 161 dibandingkan dengan model VGG16, performa model ini masih rendah. Presisi dan recall juga memiliki nilai yang rendah, dengan f1-score sebesar 10.66% sehingga model ini masih perlu ditingkatkan untuk meningkatkan kemampuan model dalam mengklasifikasikan data gambar yang dimiliki.

```

Accuracy: 0.2762096774193548
Precision: 0.10361421246939326
Recall: 0.11161321736387439
F1-score: 0.10658011360039102

```

	precision	recall	f1-score	support
0	0.16	0.15	0.15	41
1	0.00	0.00	0.00	1
2	0.03	0.02	0.02	59
3	0.09	0.10	0.09	70
4	0.00	0.00	0.00	1
5	0.46	0.49	0.47	229
6	0.07	0.11	0.08	19
7	0.06	0.07	0.06	58
8	0.00	0.00	0.00	18
accuracy			0.27	496
macro avg	0.10	0.10	0.10	496
weighted avg	0.25	0.27	0.26	496

```

Correct predictions: 335
Wrong predictions: 161

```

gambar 16. Evaluasi mobileNet Epoch 10 Pada Test Set

Setelah melakukan tuning pada model dengan menggunakan model yang sama, yaitu mobileNet dan penambahan jumlah epoch dari 10 epoch menjadi 30 epoch, terdapat sedikit penurunan akurasi menjadi 26.20%. Meskipun terdapat peningkatan dalam jumlah prediksi yang benar menjadi 343 dan prediksi yang salah menjadi 153 dibandingkan dengan model dasar, performa model ini masih rendah. Presisi, recall, dan f1-score juga masih rendah dengan nilai sebesar 10.06%.

```

Accuracy: 0.2620967741935484
Precision: 0.10137854762841172
Recall: 0.10169867488126497
F1-score: 0.10061492631417306

```

	precision	recall	f1-score	support
0	0.11	0.17	0.14	41
1	0.00	0.00	0.00	1
2	0.17	0.14	0.15	59
3	0.13	0.13	0.13	70
4	0.00	0.00	0.00	1
5	0.43	0.39	0.41	229
6	0.00	0.00	0.00	19
7	0.14	0.14	0.14	58
8	0.04	0.06	0.04	18
9	0.00	0.00	0.00	0
accuracy			0.25	496
macro avg	0.10	0.10	0.10	496
weighted avg	0.27	0.25	0.26	496

Correct predictions: 343

Wrong predictions: 153

gambar 17. Evaluasi mobileNet Epoch 30 Pada Test Set

Jika dibandingkan dengan model VGG16, model ini memiliki performa yang serupa namun dengan nilai akurasi dan f1-score yang sedikit lebih rendah, sehingga kami perlu melakukan peningkatan lebih lanjut pada kedua model ini untuk mencapai kinerja yang lebih baik.

c) DenseNet121

Pada Base Model dengan menggunakan denseNet121 dan jumlah epoch sebanyak 10 kali dapat kita lihat bahwa nilai accuracy, precision, recall, dan f1 scorenya sangat rendah hasil evaluation-nya menunjukkan bahwa model kita hanya memiliki tingkat akurasi sebesar 31.05%. Meskipun terdapat 310 prediksi yang benar dan 186 prediksi yang salah, performa model ini masih rendah. Presisi dan recall juga memiliki nilai yang rendah, dengan f1-score sebesar 9.40% sehingga model ini masih perlu ditingkatkan untuk bisa lebih baik mengklasifikasikan data.

```
Accuracy: 0.31048387096774194
Precision: 0.09325233638462649
Recall: 0.09972354528734126
F1-score: 0.09401602499350387
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	41
1	0.00	0.00	0.00	1
2	0.12	0.10	0.11	59
3	0.08	0.04	0.06	70
4	0.00	0.00	0.00	1
5	0.45	0.56	0.50	229
6	0.00	0.00	0.00	19
7	0.12	0.19	0.15	58
8	0.00	0.00	0.00	18
accuracy			0.30	496
macro avg	0.09	0.10	0.09	496
weighted avg	0.25	0.30	0.27	496

Correct predictions: 310
Wrong predictions: 186

gambar 18. Evaluasi denseNet121 Epoch 10 Pada Test Set

Setelah melakukan tuning pada model dengan menggunakan model yang sama, yaitu denseNet121 dan penambahan jumlah epoch dari 10 epoch menjadi 30 epoch, terdapat sedikit penurunan akurasi menjadi 29.84%. Meskipun terdapat peningkatan dalam jumlah prediksi yang benar menjadi 326 dan prediksi yang salah menjadi 170 dibandingkan dengan model dasar, performa model ini masih rendah. Presisi, recall, dan f1-score juga masih rendah dengan nilai sebesar 10.64%.

Accuracy: 0.29838709677419356
Precision: 0.11135313496285928
Recall: 0.10613142866366905
F1-score: 0.10638699831599503

	precision	recall	f1-score	support
0	0.09	0.10	0.09	41
1	0.00	0.00	0.00	1
2	0.09	0.05	0.06	59
3	0.02	0.01	0.02	70
4	0.00	0.00	0.00	1
5	0.47	0.56	0.51	229
6	0.00	0.00	0.00	19
7	0.13	0.16	0.14	58
8	0.13	0.11	0.12	18
9	0.00	0.00	0.00	0
accuracy			0.30	496
macro avg	0.09	0.10	0.10	496
weighted avg	0.26	0.30	0.28	496

Correct predictions: 326
Wrong predictions: 170

gambar 19. Evaluasi denseNet121 Epoch 30 Pada Test Set

3) Hasil dan Analisa

Berikut merupakan Hasil dan Analisa dari beberapa Base Model beserta Tuning-nya

Object Recognition

Model	Base/Tuning	Train dan Validation Set	Test Set
YOLOv5n	Base Model	precision: 0,338 recall: 0,0101 mAP50: 0,00195 mAP50-95: 0,00045	-

Hasil precision dan recallnya sangat kecil pada Train dan Validation Set dikarenakan epoch-nya yang terlalu sedikit, yaitu 5. Untuk men-training 5 epoch dibutuhkan sekitar 5 jam sehingga kami memutuskan untuk tidak meng-evaluasinya menggunakan test set dan lanjut menjalankan eksperimennya dengan Image Classification bukan Object Recognition lagi.

(p.s: untuk detail analisa mengenai hasilnya sudah kami jelaskan di bagian [Base Model>YOLOv5n](#))

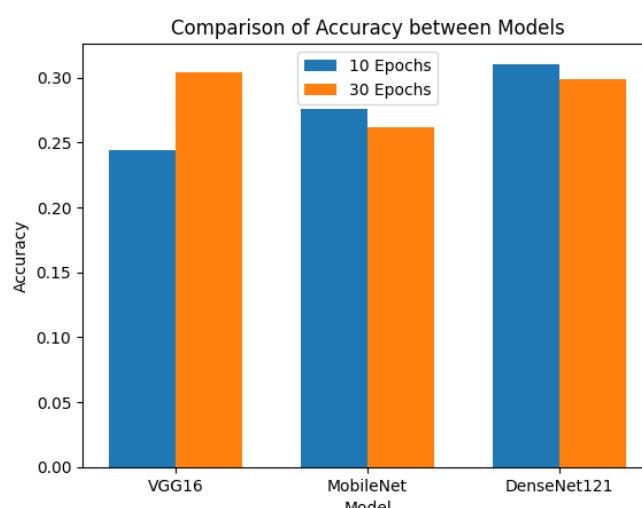
Image Classification

Model	Base/Tuning	Train dan Validation Set	Test Set
VGG16	Base Model	accuracy: 0.6222 val_accuracy: 0.5887	Accuracy: 0.24395 Precision: 0.09062 Recall: 0.09658 F1-score: 0.09230
	Tuning Model	accuracy: 0.6826 val_accuracy: 0.6310	Accuracy: 0.30443 Precision: 0.09949 Recall: 0.10708 F1-score: 0.10058
MobileNet	Base Model	accuracy: 0.7487	Accuracy: 0.27620

		val_accuracy: 0.6734	Precision: 0.10361 Recall: 0.11161 F1-score: 0.10658
	Tuning Model	accuracy: 0.8765 val_accuracy: 0.6552	Accuracy: 0.26209 Precision: 0.10137 Recall: 0.10169 F1-score: 0.10061
DenseNet121	Base Model	accuracy: 0.6965 val_accuracy: 0.6431	Accuracy: 0.31048 Precision: 0.09325 Recall: 0.09972 F1-score: 0.09401
	Tuning Model	accuracy: 0.8182 val_accuracy: 0.6431	Accuracy: 0.29838 Precision: 0.11135 Recall: 0.10613 F1-score: 0.10638

Sesuai dengan table diatas kita bisa melihat adanya perbedaan yang signifikan antara accuracy pada validation set dan test set. Perbedaan akurasi yang signifikan ini memiliki arti bahwa model yang kita miliki mengalami overfitting. Overfitting merupakan kondisi ketika model terlalu cocok dengan data yang digunakan untuk melatihnya, sehingga tidak mampu secara efektif memgeneralisasi dan memprediksi dengan akurat pada data yang belum pernah dilihat sebelumnya dalam hal ini merupakan test set.

4) Kesimpulan



gambar 20. Perbandingan Akurasi Tiap Model Terhadap Test Set

Kesimpulan yang bisa kami ambil dalam eksperimen Multiclass Image Project ini adalah model dengan accuracy tertinggi adalah model denseNet121 dengan jumlah epoch sebanyak 10 kali yang menghasilkan accuracy pada test sebesar 0.31048 atau sekitar 31.05% . Meskipun model denseNet121 dengan epoch 10 ini berhasil mencapai akurasi yang lebih tinggi dibandingkan model VGG16 dan mobileNet dengan jumlah epoch 10 dan 30, tetapi perlu diingat bahwa akurasi 31.05% masih sangatlah rendah. Dalam kondisi ini model yang kita miliki masih mengklasifikasikan gambar seperti bermain tebak-tebakan sehingga masih diperlukan pengembangan terhadap model ini dengan mengatasi permasalahan overfitting seperti menggunakan dropout, cross validation, ataupun regularisasi L1/L2.

Selain itu pengembangan terhadap model bisa juga bisa dilakukan dengan cara menambahkan jumlah data pada train set dan mengekstrak coordinate bounding box (xmin, xmax, ymin, dan ymax) sebagai fitur tambahan pada model.

5) Referensi

- <https://machinelearningmastery.com/object-recognition-with-deep-learning/>
- https://github.com/Sanansuleymanov/Custom_Infrared_Human_Detection_YOLOv5n
- <https://blog.paperspace.com/train-yolov5-custom-data/>
- <https://github.com/AarohiSingla/yolov5>
- <https://github.com/vijayg15/Keras-MultiClass-Image-Classification>
- <https://github.com/prashant-raghu/Multiclass-scene-classification-using-tf.keras>

6) Lampiran

Berikut ini kami lampirkan semua colab hasil percobaan kami, dan juga folder images & label untuk YOLO, folder yolov5 yang berisi best.pt dan last.pt, hingga file .h5 untuk setiap model image classification

[pretrained model, images & label, YOLOv5](#)