TELKOMSEL TECHNICAL TEST

Ade Dwi Arini Putri

Problem Set 1

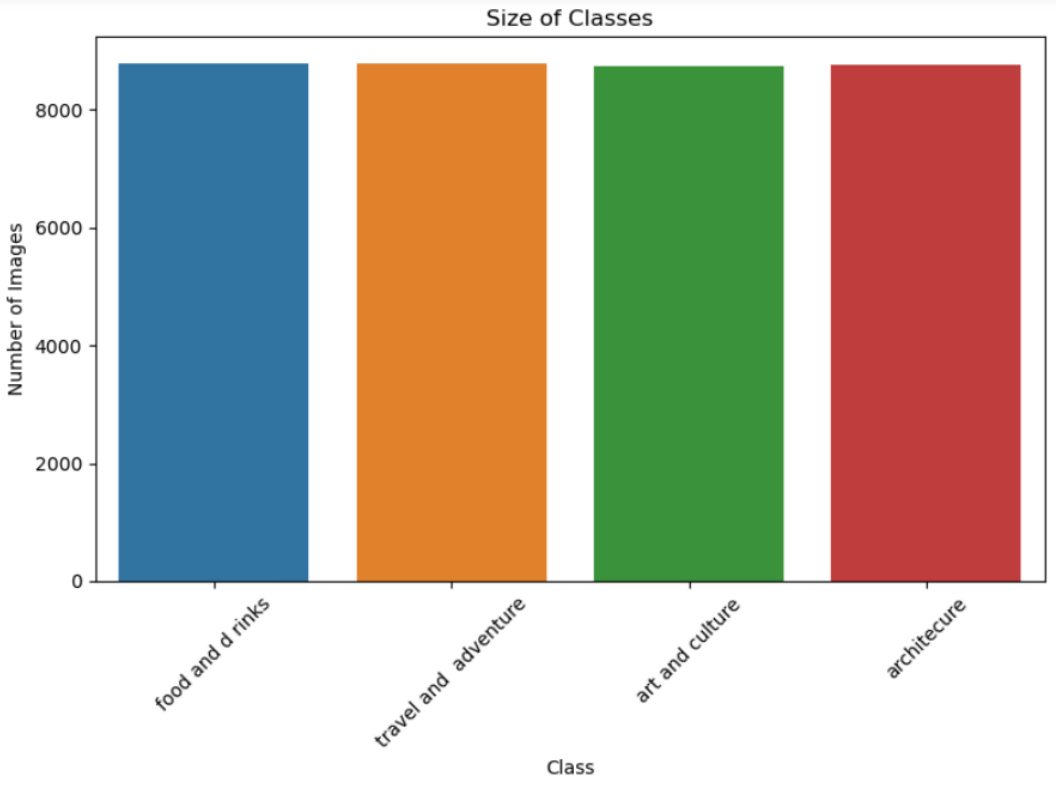
Google Image Scraped Dataset

Link : https://www.kaggle.com/datasets/duttadebadri/image-classification

Dataset from Kaggle from user DEBADRI DUTTA which was updated five years ago. The data contains images with classification: Art & Culture, Architecture, Food and Drinks, Travel and Adventure. Three folders contain images (which are used for training the dataset), tests, and validation. Each folder has the four categories mentioned above. The images file :

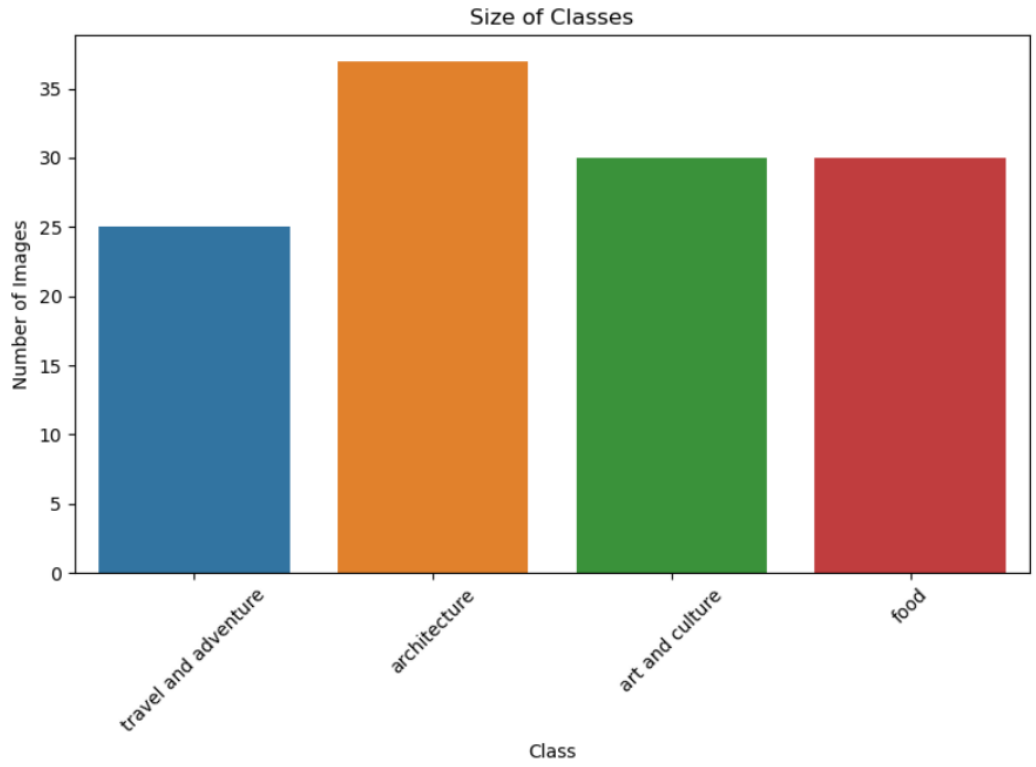
Train data

Folder Name	Dataset
architecture	8763
art and culture	8750
food and drinks	8782
travel and adventure	8800



Validation Data

Folder Name	Dataset
architecture	37
art and culture	30
food and drinks	30
travel and adventure	25

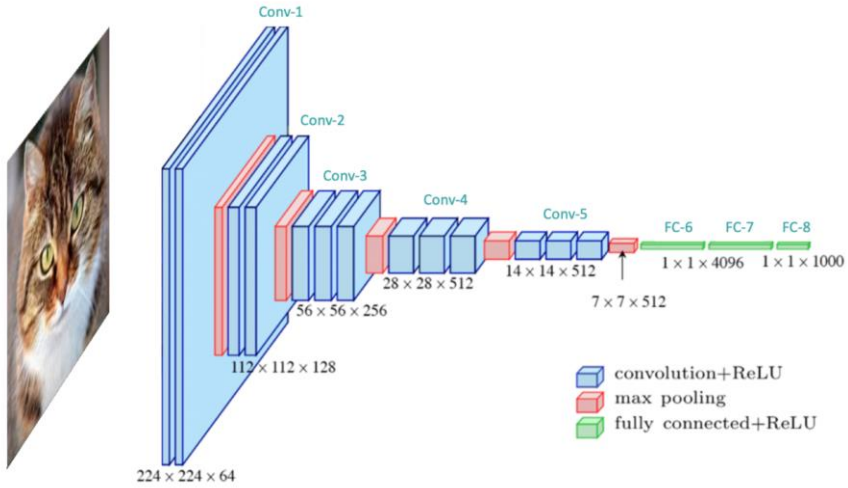


Problem Statement:

The user would like to classify the images based on its own categories.

How to solve :

Convolutional neural network (CNN) is a multilayer neural network. Convolutional Neural Networks (CNN) were created to handle picture data more effectively and efficiently. The ability of convolutional layers to process various portions of the input picture using the same weights is known as parameter sharing. As the kernel traverses over the image, this helps to identify feature patterns that are translation invariant. This method increases the model efficiency by drastically lowering the overall number of trainable parameters when compared to fully linked layers.



Source : https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

The model code will be explain one by one :

```
#import libraries
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
import os
import random
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import History
from PIL import Image
```

```
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization, Dropout
```

These are libraries that are used for this classification problem statement.

- Tensorflow is a widely used deep-learning framework for building and training neural networks.
- Seaborn is a Python data visualization library; it provides a high-level interface for creating plots for visualizing the result.
- Matplotlib is also a Python data visualization library; it provides a simple and convenient interface to help plot the result.
- The 'os' module in Python interacts with the operating system; it can perform operations of reading and writing the data.
- Random library provides a function to generate random numbers.
- NumPy supports large, multi-dimensional arrays, matrices, and collections of mathematical functions.
- Keras.models.Sequential = "Sequential" is a class from Keras, which is a linear stack of layers. It helps to build deep learning by stacking multiple layers on top of each other.
- Keras.layers : Conv2D, MaxPooling2D, Flatten, Dense, and Dropout help define the architecture and behavior of the neural network model.
- Keras.preprocessing.image.ImageDataGenerator: ImageDataGenerator is used for data augmentation and preprocessing of images. It helps perform image transformation of rescaling, rotation, shear, zoom, and horizontal flip.
- Keras.callbacks.History: The History callback that records training metrics and loss values during model training.
- PIL provides a set of image processing functions for loading and manipulating images.
- Regularizers are used to apply penalties on the model's parameters during training, which helps prevent overfitting.
- Batch normalization is a technique used to normalize the activations of a neural network layer, making the model more stable and easier to train.
- Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of the input units to zero during training.

```
# Set the path to the dataset folder
dataset_path = '/kaggle/input/image-classification/'
train_data_dir = os.path.join(dataset_path, 'images', 'images')
validation_data_dir = os.path.join(dataset_path, 'validation', 'validation')
```

The code above is used for accessing the dataset of training and validation. The main path is in dataset_path, the train path is in train_data_dir and the validation is in validation_data_dir.

```
: # Count the number of train images in each class
  class_sizes = []
  class_labels = []
  for class_label in os.listdir(train_data_dir):
      class_folder = os.path.join(train_data_dir, class_label)
      if os.path.isdir(class_folder):
          num_images = len(os.listdir(class_folder))
          class_sizes.append(num_images)
          class_labels.append(class_label)

  # Create the bar plot
  plt.figure(figsize=(8, 6))
  sns.barplot(x=class_labels, y=class_sizes)
  plt.xlabel('Class')
  plt.ylabel('Number of Images')
  plt.title('Size of Classes')
  plt.xticks(rotation=45)
  plt.tight_layout()
  plt.show()
```

The code above is used to count how many dataset in each folder of training, and then visualized them with seaborn and matplotlib.

Below is the same code, but rather is also used to visualize validation dataset.

```
# Count the number of validation images in each class
vclass_sizes = []
vclass_labels = []
for vclass_label in os.listdir(validation_data_dir):
    vclass_folder = os.path.join(validation_data_dir, vclass_label)
    if os.path.isdir(vclass_folder):
        num_images = len(os.listdir(vclass_folder))
        vclass_sizes.append(num_images)
        vclass_labels.append(vclass_label)

# Create the bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x=vclass_labels, y=vclass_sizes)
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.title('Size of Classes')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Train dataset :

```
: # Set the image dimensions
  img_width, img_height = 150, 150

  # Set the number of epochs and batch size
  epochs = 20
  batch_size = 32
```

The number of img_width and img_height to maintain a balance between capturing sufficient details of the images and computational efficiency.

The number of epochs determines how many times the model updates its weights based on the training data.

Batch size refers to the number of samples processed by the model in each training iteration before updating the weights.

```
# Generate augmented training data
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Only rescale the validation data
validation_datagen = ImageDataGenerator(rescale=1.0 / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 35093 images belonging to 4 classes.
Found 122 images belonging to 4 classes.
```

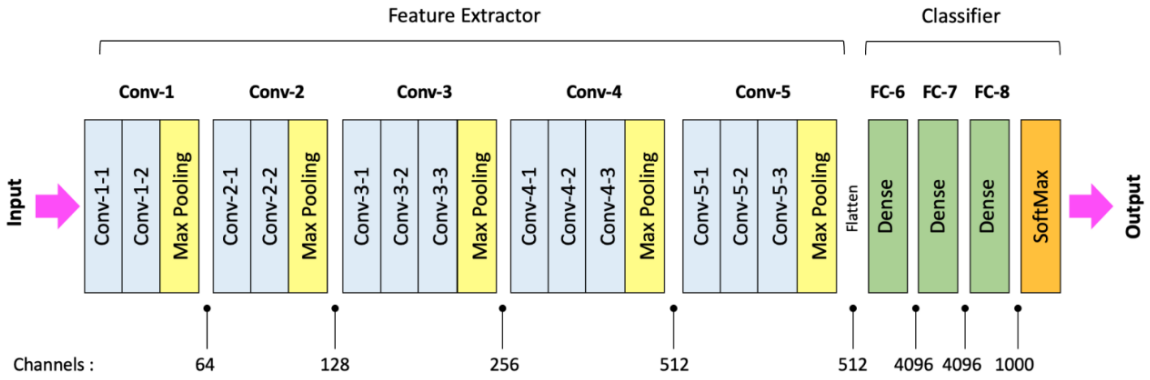
ImageDataGenerator() generated class for augmenting images of train dataset.

Performing rescale, shear, zoom and horizontal_flip to train dataset will

have an impact on the performance and accuracy of an image classification model. Which are : create data density, improved robustness, and regularization. Meanwhile augmenting data for validation is not necessary because the validation data is used to evaluate performance of the model.

Build the model

```
model = Sequential()

model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(img_width, img_height, 3)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(4, activation='softmax'))

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



As the picture above shows :

First generating the conv-1 with the layer of 64 and 3,3 size; and then performing padding=same which can help the model learn more abstract and hierarchical representations of the input data, pool size of 2,2 reduces the spatial dimensions of the input feature maps, preserving the most important features while discarding some spatial information; Drop out 0.25 prevent overfitting by reducing co-adaptation between neurons. The other three conv-network are applying the same parameters with different layers. The purpose of this is to allows the model to learn and capture different levels of features and abstractions from the input data. Which helps in depth complexity, increasing capacity and it represents hierarchical representation learning of the model.

After performing conv-network, the image will be flatten to convert the output of the previous layer, which is typically a 2D feature map, into a 1D vector.

Dense, which also known as fully connected network which will learn complex patterns by transforming the input features through a series of weighted computations. Softmax activation is being used for multi-class classification problems, where the goal is to assign an input to one of several possible classes.

The last one is compile, its prepares the model for the training process. With the optimizer adam, accuracy metric and loss categorical_crossentropy.

```
#execute training process
history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator
)
```

This is execute the model with the parameters that was defined above.