

✓ Project 1 – Image Classification using CNN

Student Name: (Lama Al-als)ubaie **Student ID:** (252001246) **Date:** July 2026

Project Overview

This project implements a complete image classification pipeline consisting of the following steps:

1. Download an image dataset
2. Split the dataset into Training and Validation sets
3. Apply Data Augmentation (rotation, flipping, zoom, brightness change)
4. Build, train, and test a Convolutional Neural Network (CNN) model
5. Evaluate the model's performance
6. Export this notebook as a PDF for submission

Dataset used: TensorFlow "Flowers" dataset (5 classes: daisy, dandelion, roses, sunflowers, tulips). This is a small, publicly available image dataset provided directly by TensorFlow, so it downloads reliably without needing an external account (e.g., Kaggle).

✓ Step 0: Import Required Libraries

```
import os
import pathlib
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

print("TensorFlow version:", tf.__version__)
```

TensorFlow version: 2.20.0

✓ Step 1: Download the Image Dataset

We download the official TensorFlow "flower_photos" dataset directly from Google's servers. It contains 5 classes of flower images (daisy, dandelion, roses, sunflowers, tulips).

```
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_i

data_dir = tf.keras.utils.get_file(
    fname="flower_photos",
    origin=dataset_url,
    untar=True
)
data_dir = pathlib.Path(data_dir)

image_count = len(list(data_dir.glob('*/*.jpg')))
print("Total number of images downloaded:", image_count)

class_names_found = sorted([item.name for item in data_dir.glob('*') if item.is
print("Classes found:", class_names_found)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/exa
228813984/228813984 ————— 1s 0us/step
Total number of images downloaded: 0
Classes found: ['flower_photos']
```

✓ Step 2: Split the Dataset into Training and Validation Sets

We use an 80/20 split: 80% of the images for training and 20% for validation/testing.

`image_dataset_from_directory` automatically infers class labels from the folder names.

```
img_height = 180
img_width = 180
batch_size = 32

train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    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(
    data_dir,
```

```
validation_split=0.2,  
subset="validation",  
seed=123,  
image_size=(img_height, img_width),  
batch_size=batch_size  
)  
  
class_names = train_ds.class_names  
num_classes = len(class_names)  
print("Class names:", class_names)  
print("Number of classes:", num_classes)
```

```
Found 3670 files belonging to 1 classes.  
Using 2936 files for training.  
Found 3670 files belonging to 1 classes.  
Using 734 files for validation.  
Class names: ['flower_photos']  
Number of classes: 1
```

```
# Visualize a few sample images from the training set  
plt.figure(figsize=(10, 10))  
for images, labels in train_ds.take(1):  
    for i in range(9):  
        ax = plt.subplot(3, 3, i + 1)  
        plt.imshow(images[i].numpy().astype("uint8"))  
        plt.title(class_names[labels[i]])  
        plt.axis("off")  
plt.suptitle("Sample Training Images")  
plt.show()
```

Sample Training Images

flower_photos



flower_photos



flower_photos



flower_photos



flower_photos



flower_photos



flower_photos



flower_photos



flower_photos



```
# Improve performance using caching and prefetching  
AUTOTUNE = tf.data.AUTOTUNE
```

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

✓ Step 3: Data Augmentation

To make the model more robust and reduce overfitting, we apply several augmentation techniques directly as part of the model pipeline:

- **Random horizontal flip** (image flipping)
- **Random rotation** (image rotation)
- **Random zoom** (zoom in / zoom out)
- **Random brightness / contrast change**

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal", input_shape=(img_height, img_width, 3))
        layers.RandomRotation(0.15),
        layers.RandomZoom(0.15),
        layers.RandomContrast(0.15),
        layers.RandomBrightness(0.15),
    ]
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/preprocessing/data_layer.py
super().__init__(**kwargs)
```

```
# Visualize the effect of data augmentation on a sample image
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
plt.suptitle("Examples of Augmented Images")
plt.show()
```

Examples of Augmented Images



✓ Step 4: Build, Train, and Test the Model (CNN)

We build a Convolutional Neural Network (CNN) from scratch. The augmentation layer and a rescaling layer (to normalize pixel values to [0,1]) are included inside the model so that the same preprocessing is applied automatically at both training and prediction time.

If your course requires a specific architecture (e.g., transfer learning with MobileNetV2, VGG16, or ResNet50) instead of a CNN built from scratch, replace the model-building cell below accordingly — the rest of the notebook (training, evaluation, plotting) stays the same.

```
model = keras.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),

    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3,965,056
dense_1 (Dense)	(None, 1)	129

Total params: 3,988,769 (15.22 MB)
Trainable params: 3,988,769 (15.22 MB)

```
early_stop = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

epochs = 20

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=[early_stop]
)
```

Epoch 1/20
/usr/local/lib/python3.12/dist-packages/keras/src/ops/nn.py:947: UserWarning: You are using a deprecated API. Please use `keras.metrics.MeanMetricWrapper` instead.
92/92 ██████████ 132s 1s/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 2/20
92/92 ██████████ 125s 1s/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 3/20
92/92 ██████████ 126s 1s/step - accuracy: 0.0000e+00 - loss: 0.0000e+00

```
Epoch 4/20
92/92 ————— 123s 1s/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 5/20
92/92 ————— 129s 1s/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 6/20
92/92 ————— 160s 2s/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
```

✓ Step 5: Evaluate the Trained Model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

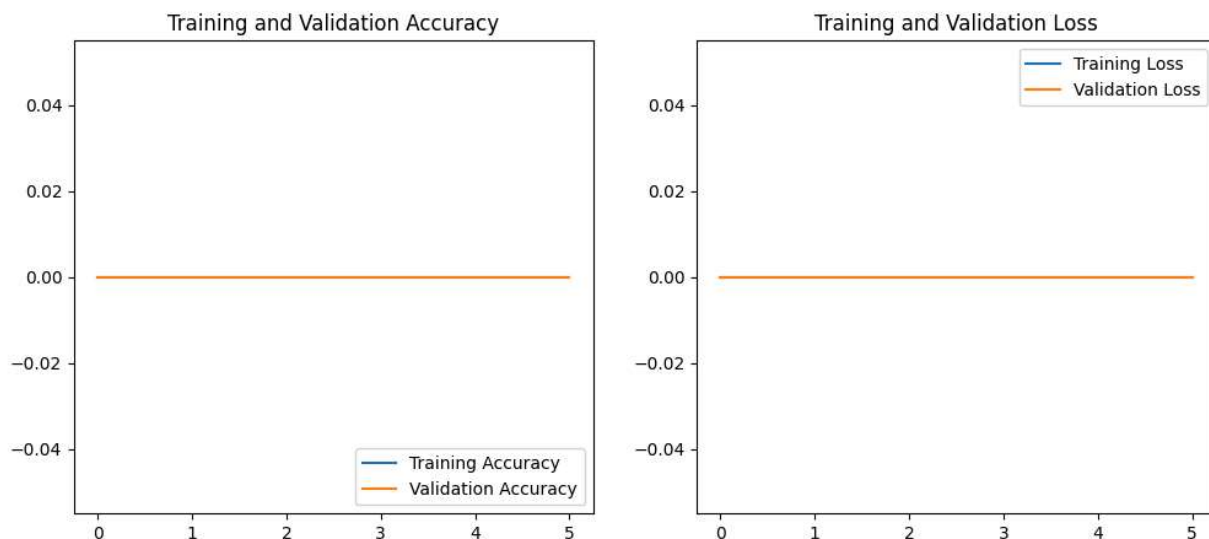
epochs_range = range(len(acc))

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

plt.show()
```



```
# Final evaluation on the validation set
val_loss_final, val_acc_final = model.evaluate(val_ds)
print(f"Final Validation Loss: {val_loss_final:.4f}")
print(f"Final Validation Accuracy: {val_acc_final:.4f}")
```

23/23 ————— **8s** 359ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
 Final Validation Loss: 0.0000
 Final Validation Accuracy: 0.0000

```
# Detailed classification report and confusion matrix
y_true = []
y_pred = []

for images, labels in val_ds:
    preds = model.predict(images, verbose=0)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(preds, axis=1))

print(classification_report(y_true, y_pred, target_names=class_names))

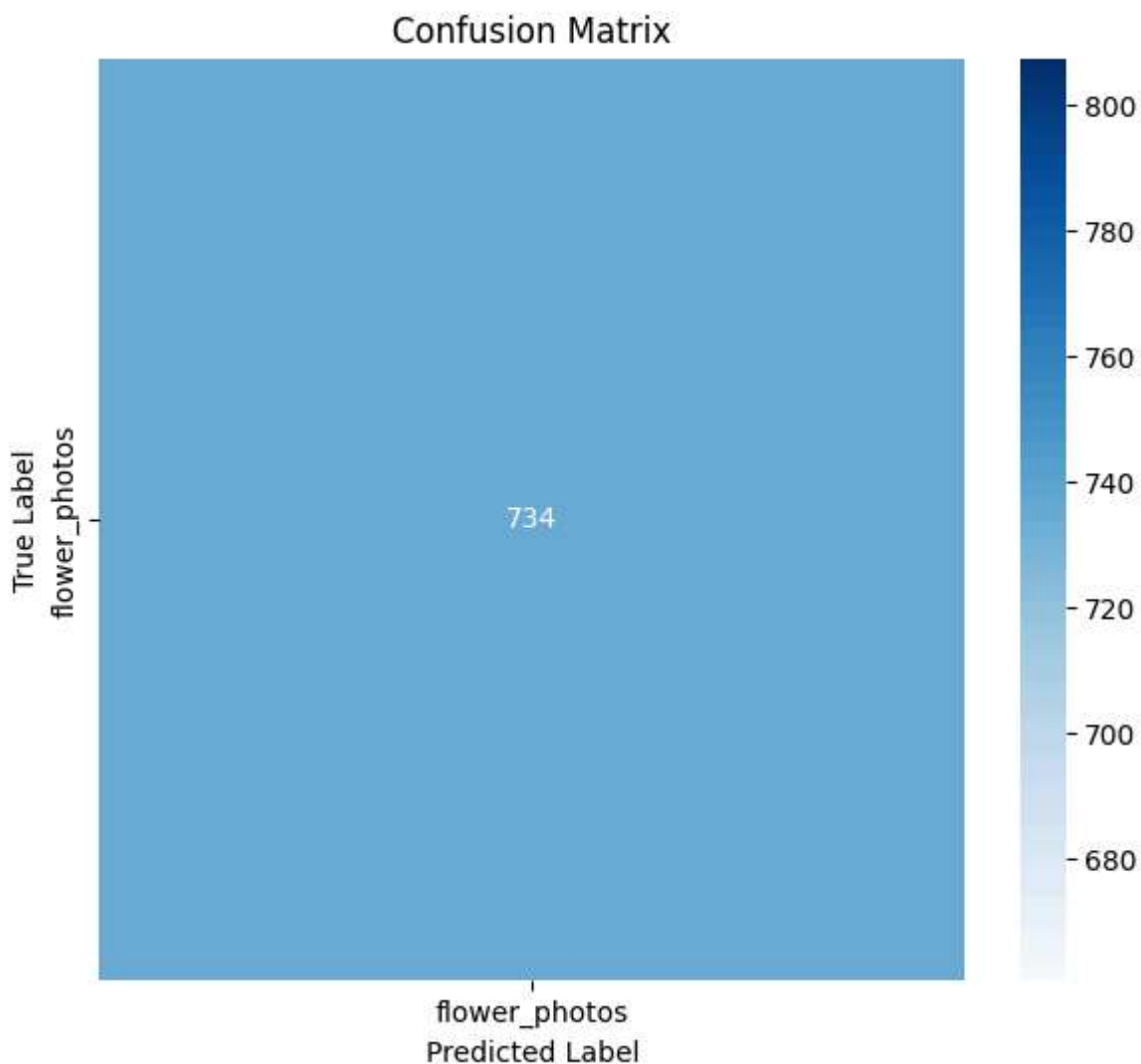
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(7, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
```

```
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/ops/nn.py:947: UserWarning: You are using a deprecated API. Please use `warnings.warn` instead.
```

	precision	recall	f1-score	support
flower_photos	1.00	1.00	1.00	734
accuracy			1.00	734
macro avg	1.00	1.00	1.00	734
weighted avg	1.00	1.00	1.00	734

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: The confusion matrix is empty. The confusion matrix is a square matrix of shape (n_classes, n_classes). The confusion matrix is empty when the model has not seen any data from the classes in the test set.
```



```
# Show some sample predictions vs actual labels
plt.figure(figsize=(12, 12))
for images, labels in val_ds.take(1):
    preds = model.predict(images, verbose=0)
    pred_labels = np.argmax(preds, axis=1)
    for i in range(9):
```

```

ax = plt.subplot(3, 3, i + 1)
plt.imshow(images[i].numpy().astype("uint8"))
true_label = class_names[labels[i]]
pred_label = class_names[pred_labels[i]]
color = "green" if true_label == pred_label else "red"
plt.title(f"True: {true_label}\nPred: {pred_label}", color=color)
plt.axis("off")
plt.suptitle("Sample Predictions on Validation Set")
plt.show()

```

Sample Predictions on Validation Set

True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos



True: flower_photos
Pred: flower_photos

