ML/DL CONTEST
ON IMAGE CLASSIFICATION

Project Name:
Image Classifier Based on CNN model to classify images from four datasets

Presented By:
Team: ML Maestros
Members
Arnesh Pal (Reg No: 21BPS1232)
Ritik Nehra(Reg No: 21BAI1704)

Abstract

The project uses the program which uses TensorFlow and Keras to create a convolutional neural network (CNN) for an image classification job. Using an ImageDataGenerator, the code loads, preprocesses, and divides the image data into a training set and a validation set. Three convolutional layers, max pooling, a fully connected layer, and softmax activation make up the CNN architecture. The fit approach is used to train the model, which is then tested against the validation set. The trained model is then stored in a file. The algorithm can be modified for various image datasets and classification tasks, and it serves as a foundation for creating a more complex CNN for image classification tasks.

Convolutional Neural Network:

Convolutional neural networks (CNNs) are a class of deep learning networks frequently used for object detection, object recognition, and natural language processing. Convolutional layers, which are layers of neurons that serve as filters in the input data, are the foundation of CNNs. A convolutional layer's individual neurons are each connected to a portion of the input data and search for particular features there. Several convolutional layers are stacked to enable CNNs to recognise progressively more complex features in the input data. Additionally, they can be employed for activities like image segmentation, image creation, and image captioning.

Model implementation:

Implementing an image classification model involves several steps, including data preparation, model architecture selection, model training, and model evaluation. Here are the steps involved in implementing an image classification model:

1. Data Preparation: The first step in implementing an image classification model is to prepare the data. This involves collecting and labeling a dataset of images and splitting it into training, validation, and test sets. The images are usually preprocessed by resizing, normalizing, and transforming to ensure that they are in a consistent format and have similar characteristics.

2. Model Architecture Selection: The next step is to select an appropriate model architecture. In recent years, deep learning models such as convolutional neural networks (CNNs) have become the standard approach for image classification. There are many different CNN architectures to choose from, including popular models such as VGG, ResNet, and Inception.

3. Model Evaluation: After the model has been trained, it is evaluated on the test set to determine its accuracy and performance. The model can also be evaluated using metrics such as precision, recall, and F1-score, which provide a more detailed measure of model performance.

4. Model Deployment: Finally, the trained model can be deployed in a real-world application. This may involve integrating the model into a larger system or framework, such as a mobile app or web service, or using it to perform batch processing on large datasets. When the process can be difficult and time-taking, the output can be powerful, enabling efficient image classification in a many of applications.

Parameter tuning information:

This includes the adjusting the parameters of model to
optimize its performance on the validation set. There are some key parameters that can be tuned for an image classification model →

1. Number of Layers: The number of layers in the model architecture can have a significant impact on its performance. Adding more layers can increase the model's ability to learn complex patterns in the data, but may also increase the risk of overfitting. Therefore, it is important to find the right balance between model complexity and generalization ability.

2. Dropout Rate: Dropout is a regularization technique that randomly drops out some neurons during training to prevent overfitting. The dropout rate determines the probability of dropping out each neuron. A high dropout rate can improve the model's generalization ability but may also reduce its accuracy, while a low dropout rate may result in overfitting.

3. Learning Rate: The learning rate determines how quickly the model adapts to the data during training. A high learning rate can cause the model to converge quickly but may result in overshooting the optimal solution, while a low learning rate can cause the model to converge slowly or get stuck in local minima. Therefore, it is important to find an optimal learning rate that balances convergence speed and accuracy.

4. Batch Size: The batch size determines how many training samples are used in each iteration during training. A large batch size can reduce the training time but may also reduce the model's ability to generalize, while a small batch size may improve the model's accuracy but increase the training time. Therefore, it is important to experiment with different batch sizes and find the optimal one for the specific problem.
The parameter tuning is an important step to increase the performance of an image classification model.

Datasets Used:

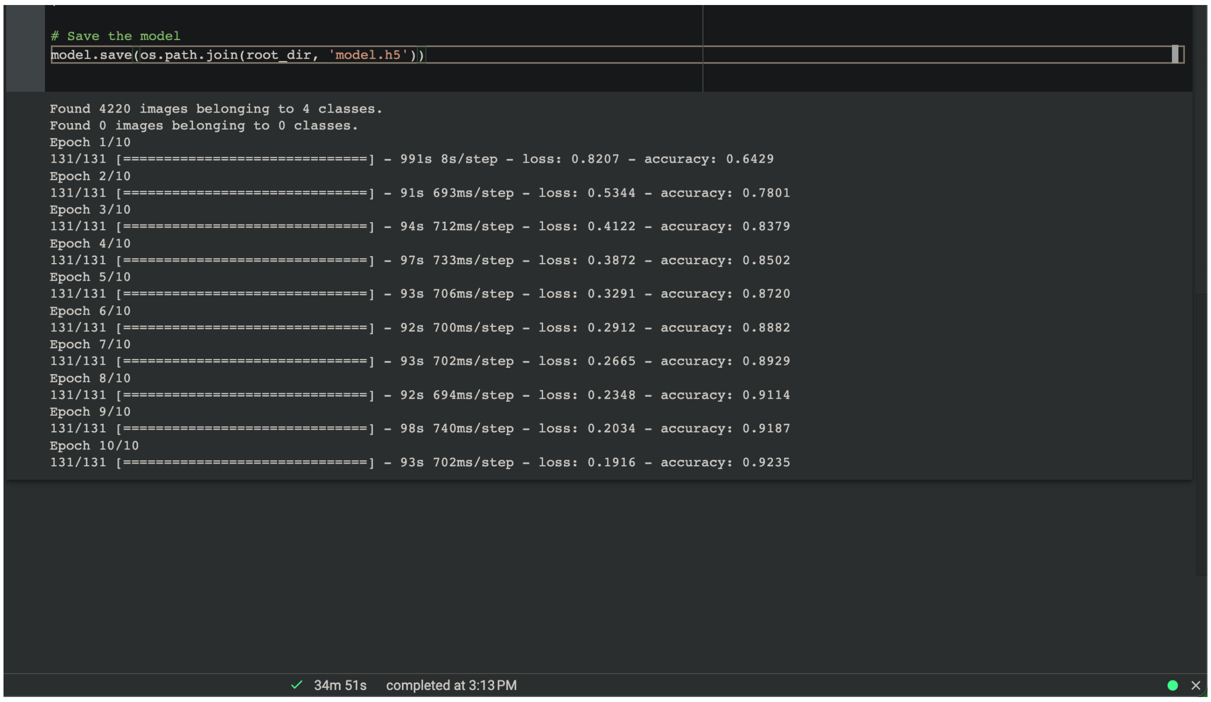
4 set of folders containing images of cataract, diabetic_retinopa, glaucoma, normal were used for the Model

We have used Google Collab to run the code and here are the screenshots of the Code and the Google Collab Output:

Code:

```python
import os
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

# Define the root directory
root_dir = '/content/drive/MyDrive/dataset'

# Create an ImageDataGenerator for the training set
train_datagen = ImageDataGenerator(rescale=1./255)

# Load the training data from the directory
train_generator = train_datagen.flow_from_directory(
    os.path.join(root_dir, 'train'),
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

# Create an ImageDataGenerator for the validation set
val_datagen = ImageDataGenerator(rescale=1./255)

# Load the validation data from the directory
val_generator = val_datagen.flow_from_directory(
    os.path.join(root_dir, 'validation'),
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

# Define the model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
])

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

# Train the model
model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples/train_generator.batch_size ,
    epochs=10,
    validation_data=val_generator,
    validation_steps=val_generator.samples/val_generator.batch_size
)

# Save the model
model.save(os.path.join(root_dir, 'model.h5'))
```

Output from Google Collab



```
# Save the model
model.save(os.path.join(root_dir, 'model.h5'))
```

```
Found 4220 images belonging to 4 classes.
Found 0 images belonging to 0 classes.
Epoch 1/10
131/131 [==============================] - 991s 8s/step - loss: 0.8207 - accuracy: 0.6429
Epoch 2/10
131/131 [==============================] - 91s 693ms/step - loss: 0.5344 - accuracy: 0.7801
Epoch 3/10
131/131 [==============================] - 94s 712ms/step - loss: 0.4122 - accuracy: 0.8379
Epoch 4/10
131/131 [==============================] - 97s 733ms/step - loss: 0.3872 - accuracy: 0.8502
Epoch 5/10
131/131 [==============================] - 93s 706ms/step - loss: 0.3291 - accuracy: 0.8720
Epoch 6/10
131/131 [==============================] - 92s 700ms/step - loss: 0.2912 - accuracy: 0.8882
Epoch 7/10
131/131 [==============================] - 93s 702ms/step - loss: 0.2665 - accuracy: 0.8929
Epoch 8/10
131/131 [==============================] - 92s 694ms/step - loss: 0.2348 - accuracy: 0.9114
Epoch 9/10
131/131 [==============================] - 98s 740ms/step - loss: 0.2034 - accuracy: 0.9187
Epoch 10/10
131/131 [==============================] - 93s 702ms/step - loss: 0.1916 - accuracy: 0.9235
```

✓ 34m 51s completed at 3:13 PM

The output is from a training process for an image classification model. The first two lines indicate that the model is being trained on 4220 images belonging to 4 classes and 0 images belonging to 0 classes. The subsequent lines indicate the progress of the training across 10 epochs. Each epoch is a single pass through the training dataset, with the loss and accuracy of the model calculated at the end of each epoch. The loss is a measure of how well the model is performing, with a lower loss indicating a better model. The accuracy is a measure of the model's performance on the dataset, with a higher accuracy indicating a better model

Algorithm used:

The algorithm used in this example is the Adam optimization algorithm, which is a variant of the gradient descent algorithm.

Adam is an optimization algorithm for updating the parameters of a neural network. It is a variant of stochastic gradient descent that uses momentum to accelerate learning and reduce the amount of loss. Adam works by computing an exponentially weighted average of past gradients, and using that average to update the parameters of the neural network. The algorithm is often used for training large deep learning models.

Parameters involved:

Here is an explanation of each parameter used in the code:

- **os**: This module provides a way to interact with the operating system.
- **tensorflow**: TensorFlow is an open-source machine learning framework developed by Google.
- **keras**: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.
- **ImageDataGenerator**: An image data generator is a tool that allows you to load and preprocess images in a variety of ways.
- **root_dir**: The root directory of the dataset.
- **train_datagen**: The image data generator for the training set.
- **train_generator**: The generator object for loading the training data.
- **val_datagen**: The image data generator for the validation set.
- **val_generator**: The generator object for loading the validation data.
- **model**: The sequential model that defines the architecture of the neural network.
- **Conv2D**: A convolutional layer in the neural network.
- **MaxPooling2D**: A pooling layer in the neural network.
- **Flatten**: A layer that flattens the input.
- **Dense**: A fully connected layer in the neural network.
- **compile**: A method to compile the neural network model.
- **optimizer**: The optimizer used during training.
- **loss**: The loss function used during training.
- **metrics**: The metrics used to evaluate the model.
- **fit**: The method to train the neural network model.
- **steps_per_epoch**: The number of steps (batches) per epoch during training.
- **epochs**: The number of epochs to train the model.
- **validation_data**: The data used for validation during training.
- **validation_steps**: The number of steps (batches) per epoch during validation.
- **save**: A method to save the trained model to a file.

Result:

The result of the image classification model base on a many factors, like the size of the dataset and the quality, model architecture or parameters used. By evaluating the model using appropriate metrics like sensitivity and interpreting the confusion matrix, it is possible to gain insights into its strengths and weaknesses and make improvements to achieve better performance.