

# Neural Network Project

## “ Image Classification ”

### Data Preparation Steps :

#### 1. Loading and Normalizing Images :

- Images from the training dataset are loaded and resized to (100, 100) pixels
- The pixel values are normalized to the range ]1 ,0[

```
img = load_img(img_path, target_size=(100, 100))  
img_array = img_to_array(img)  
img_array /= 255.0
```

#### 2. Labeling and Categorization :

- Labels are assigned to images based on their respective directories.
- LabelEncoder is used to convert string labels into numeric labels.
- One-hot encoding is applied to the numeric labels.

```
label_encoder = LabelEncoder()  
labels = label_encoder.fit_transform(labels)  
labels = to_categorical(labels)
```

#### 3. Data Augmentation :

- Random transformations (rotation, shift, shear, zoom, flip) are applied to original images.
- Augmented images are stored in separate lists.

```
additional_augmented_img = ImageDataGenerator(  
    rotation_range=20,  
    width_shift_range=0.4,  
    height_shift_range=0.4,  
    shear_range=0.4,  
    zoom_range=0.4,  
    horizontal_flip=True,  
    fill_mode='nearest'  
).random_transform(img)
```


#### 4. Extending Training Data :

- Augmented images and their original labels are added to the training dataset.

```
images = np.concatenate((images, additional_augmented_data),  
    axis=0)  
labels = np.concatenate((labels, additional_augmented_labels),  
    axis=0)
```

#### 5. Train-Validation Split :


- The dataset is split into training and validation sets.



```
X_train, X_val, y_train, y_val = train_test_split(images, labels,  
test_size=0.3, random_state=42)
```

## 6. Loading and Preprocessing Test Data :

- Test images are loaded and resized to (100, 100) pixels.
- Pixel values are normalized.
- Test labels are encoded and one-hot encoded.



```
img = load_img(items_path, target_size=(100, 100))  
img_array = img_to_array(img)  
img_array /= 255.0
```

## Models Architecture :

### 1. Convolutional Neural Network (CNN) :

- Sequential model with convolutional layers, max-pooling layers, batch normalization, dense layers, and dropout layers.

- The architecture consists of four convolutional layers with decreasing filter sizes and max-pooling after each convolution.
- Batch normalization is applied after each max-pooling layer.
- The flattened output is connected to two dense layers with ReLU activation and dropout for regularization.
- The final output layer has softmax activation for multi-class classification.

```
model = models.Sequential()
model.add(layers.Conv2D(256, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
# ... (Additional convolutional layers, max-pooling, batch normalization, etc.)
model.add(layers.Dense(num_classes, activation='softmax'))
```

## 2. Model Compilation & Training :

- Adam optimizer with a learning rate of 0.0001 is used.
- Categorical crossentropy is chosen as the loss function.
- Accuracy is chosen as the metric for evaluation.
- The model is trained using the training dataset with early stopping to prevent overfitting.
- Training is performed for 100 epochs with a batch size of 32.

```
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=
['accuracy'])
ThisModel = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping],
    verbose=1
)
```

## Vision Transformer Model ( ViT ) :

### 1. Model Architecture :

- Patching Layer: Applies convolution to split the input image into patches of size  $\text{patch\_size} \times \text{patch\_size}$  with a stride of  $\text{patch\_size}$

- Positional Encoding: Embeds positional information into the patches using an embedding layer.
- Multi-Head Attention: focus on different parts of the input.
- Residual Connection: Adds the original input to the attention output
- Layer Normalization: Normalizes the output to stabilize training
- Feedforward Network: Applies a feedforward neural network to capture complex patterns
- Positional Encoding Update: Updates the positional\_encoded\_patches for the next iteration.
- Global Average Pooling: Aggregates the information from all patches into a fixed-size vector by taking the average
- Classifier: A dense layer with softmax activation for classification. Create Model: Instantiates the ViT model using the defined inputs and outputs

```
def create_vit_model(image_size, num_classes, patch_size, embedding_dim,
num_heads, mlp_dim, dropout_rate):
    # Step 1: Input Layer
    inputs = layers.Input(shape=image_size)

    .....

    # Step 7: Create Model
    model = models.Model(inputs=inputs, outputs=outputs)
    return model

# Step 8: Instantiate the ViT model
vit_model = create_vit_model(image_size, num_classes, patch_size,
embedding_dim, num_heads, mlp_dim, dropout_rate)
go(f, seed, [])
}
```

## 2. Model Compilation & Training :

- compiles the model, specifying the optimizer, loss function, and metrics
- Preprocesses the data and trains the Vision Transformer model

- prints the details of the model, including layer information and parameter counts
- Evaluates the trained model on the test set and prints the accuracy
- Makes predictions on the test set and converts one-hot encoded predictions to class labels

```

optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
early_stopping = EarlyStopping(monitor='val_accuracy', patience=20,
restore_best_weights=True)
.....
vit_model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
# Assuming X_train, X_test, y_train, y_test are defined
.....
X_train = np.array(X_train).reshape(-1, 224, 224, 3)
X_test = np.array(X_test).reshape(-1, 224, 224, 3)

vit_model.fit(X_train, y_train, epochs=100, batch_size=64,
callbacks=early_stopping, validation_split=0.1)
y_pred_onehot = vit_model.predict(X_test)
y_pred_labels = np.argmax(y_pred_onehot, axis=1) + 1
y_true_classes = np.argmax(y_test, axis=1) + 1
.....

```

## Another Model ( ResNet50 ) :

### 1. Base ResNet50 Model:

- ResNet50 is used as a base model for feature extraction.
- The pre-trained weights are loaded .



```
input_shape = (224, 224, 3)
num_classes = 5

base_model = ResNet50(weights=None, include_top=False, input_shape=input_shape)

weights_path = '/kaggle/input/weights-of-
resnet50/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'
base_model.load_weights(weights_path)
```

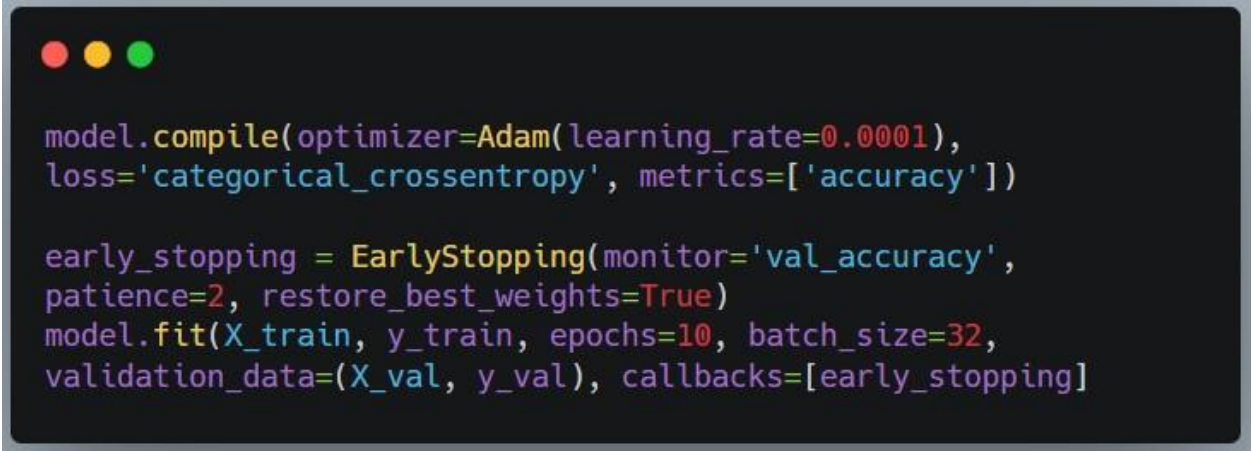
## 2. Custom Model Head:

- A Sequential model is constructed by stacking the base ResNet50 model with additional layers.
- The base model is followed by a Flatten layer, a Dense layer with ReLU activation, a Dropout layer, and a final Dense layer with softmax activation for classification.

```
model = Sequential([
    base_model,
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.8),
    Dense(num_classes, activation='softmax')
])
```

## 3. Model Compilation and Training:

- The model is compiled with a lower learning rate using the Adam optimizer.
- Early stopping is applied during training to prevent overfitting.



```
model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_accuracy',
patience=2, restore_best_weights=True)
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_val, y_val), callbacks=[early_stopping])
```

## Different Trials & Results :

### 1. CNN Model , First Trial :

- The model was trained for 100 epochs with early stopping.
- The model's accuracy on the validation set improved gradually, reaching 68.33% at the end.

However, the final test accuracy was 36.00%

### 2. CNN Model , Second Trial :

- The model was trained for 100 epochs with early stopping.
- The model's accuracy on the validation set improved gradually, reaching



76.65% at the end.

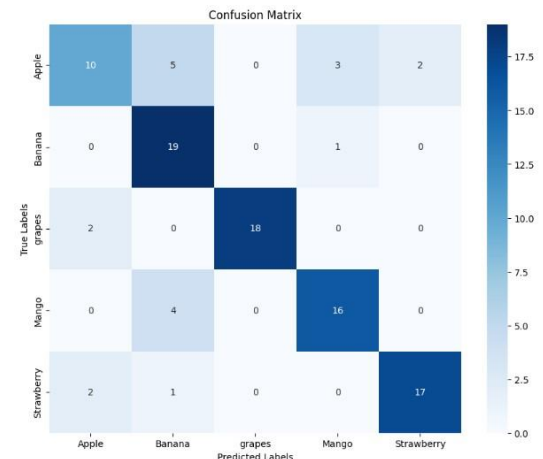
However, the final test accuracy was 69.00%

## 2. CNN Model , Third Trial :

- The model was trained for 100 epochs with early stopping.
- The model's accuracy on the validation set improved gradually, reaching 92.29% at the end.

However, the final test accuracy was 80.00%

**Test Accuracy: 80.00%**



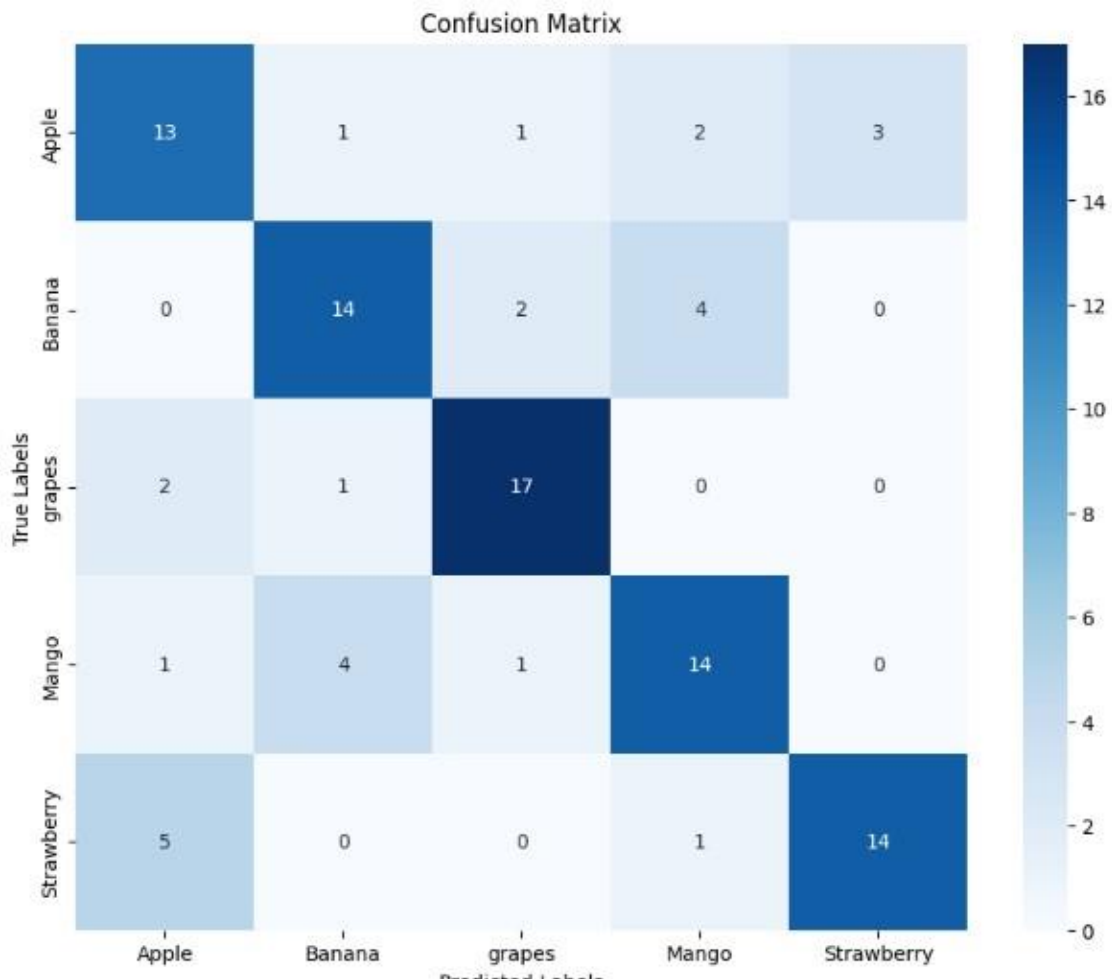
( Every Trial we changed the values of parameters , number of Layers , etc)

## 2. ViT Model:

- The ResNet50-based model was trained for 100 epochs with early stopping.
- The Training accuracy gradually increases over the epochs and reaches a peak of approximately 87.08 %

However, the final test accuracy was 72.00 %

**Test Accuracy: 72.00%**



### 3. ResNet50 Model:

#### 1. ResNet50 Model , First Trial :

- The model was trained for 10 epochs
- The model's accuracy on the validation set improved gradually, reaching 86.87% at the end.

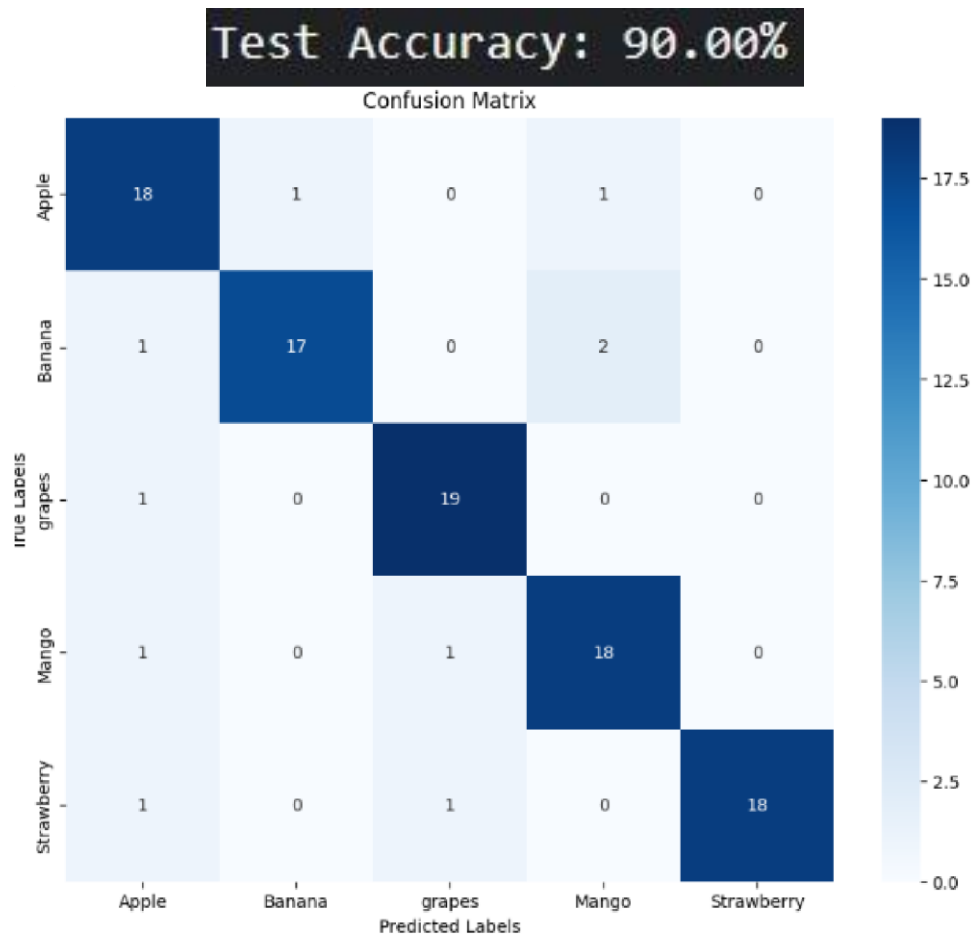
However, the final test accuracy was 36.00%

#### 2. ResNet50 Model , Second Trial :

- The model was trained for 10
- The model's accuracy on the validation set improved gradually, reaching

95.53% at the end.

However, the final test accuracy was 90.00%



### 3. Summary & Conclusion:

#### 1. CNN Model:

- Significant improvement was achieved, reaching a validation accuracy of 92.29 %, and a final test accuracy of 80.00 %

#### 2. Vision Transformer (ViT) Model:

- The ViT model, demonstrated consistent improvement during training, achieving a peak training accuracy of approximately 87.08% and the final test accuracy was 72.00 %.

#### 3. ResNet50 Model:

- Extended training (10 epochs) led to substantial improvement with a validation accuracy of 95.53 % and a higher final test accuracy of 90.00 %.

## Conclusion :

- CNN models displayed sensitivity to architecture and hyperparameter changes, showcasing a significant impact on performance.
- ResNet50 demonstrated superior performance compared to the CNN models, especially with extended training in the second trial.
- The Vision Transformer model, while achieving a relatively high training accuracy, showed a slight performance gap in the test set compared to ResNet50.