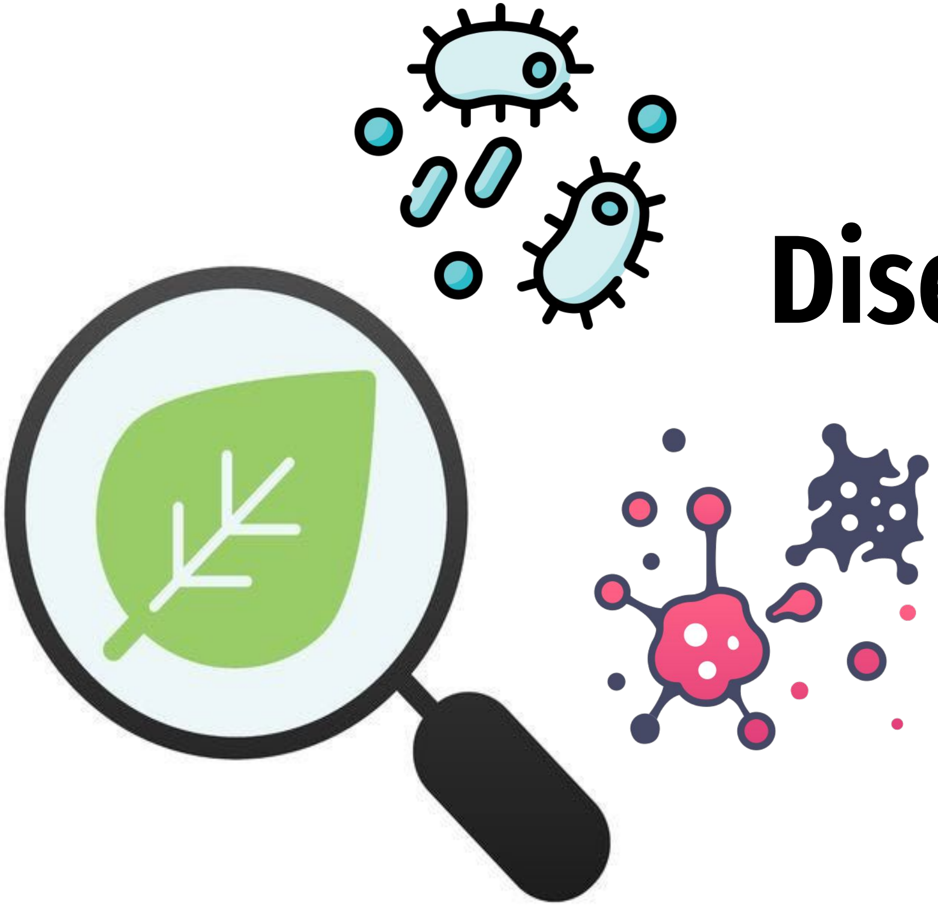


Automated Plant Disease Classification Using Machine Learning and Deep Learning

Yunseo Heo, Kelly Ng

Table of Contents

01 **Motivation**
Overview of the need of automated plant disease classification.

02 **Approach**
Outline of the project's research design and analytical techniques.

03 **Method**
Technologies and models used in the project.

04 **Results**
Analysis of each model's accuracy and comparison of their effectiveness in classifying plant diseases.

05 **Evaluation & Insight**
Insights on potential improvements, shortcomings, and areas for future research.

Motivation

Agricultural Impact

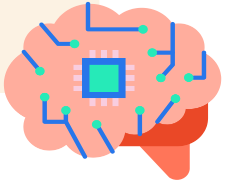
Need for Rapid Diagnosis: Early and accurate identification of plant diseases is important to prevent widespread crop damage.

Sustainability in Agriculture: Efficient disease management helps maintain ecological balance.

Technological Advancement

Leveraging Machine Learning: Use advanced algorithms to process and analyze image data to speed up the diagnosis process compared to traditional methods.

Deep Learning Potential: Exploring the capabilities of Convolutional Neural Networks (CNNs) to enhance classification accuracy.



Approach

- Develop and evaluate traditional machine learning models for plant disease classification.
- Implement deep learning models using CNNs for high-accuracy disease classification from plant images.
- Compare the performance of traditional machine learning techniques with transfer learning models to determine the most effective methods for plant disease classification.
    - Simple CNN vs transfer learning models

Bacteria

Fungi

Healthy

Pests

Virus

Methodological Framework

Collect the data

Source: Kaggle's Plant Pathogen Dataset

Prepare the data

Splitted raw data into training and validation sets and shuffled the training set

Train a model

Convolutional Neural Networks (CNN)

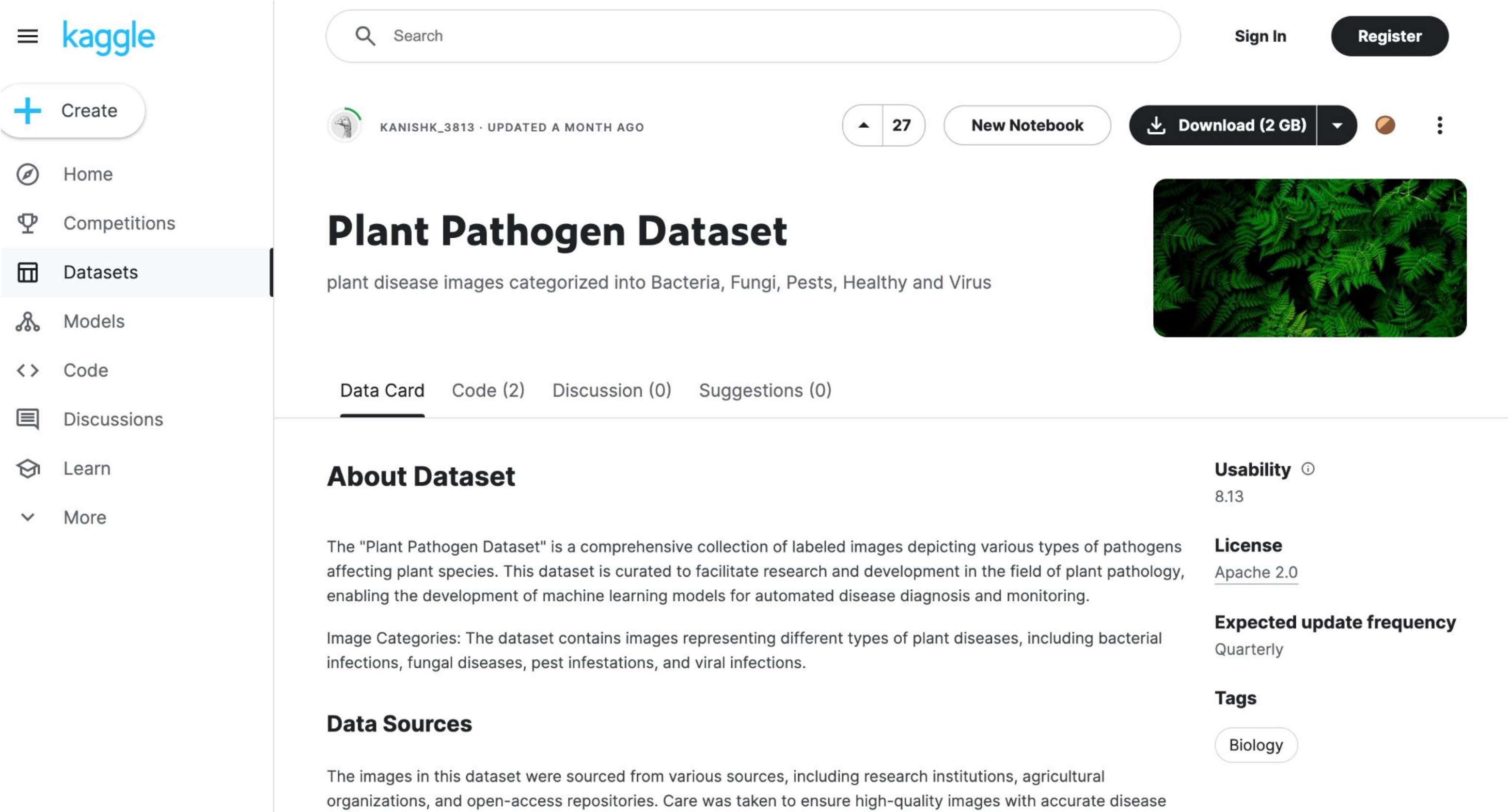
Evaluate the model

Metrics: Accuracy

Conclude

Identified the highest accuracy model

Data Acquisition



Data Preparation

```
base_path = '/Users/dbstj0426/Desktop/pathogen'

# Load the data
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_generator = datagen.flow_from_directory(base_path,
                                              target_size=(128, 128),
                                              batch_size=32,
                                              class_mode='categorical',
                                              subset='training')
val_generator = datagen.flow_from_directory(base_path,
                                            target_size=(128, 128),
                                            batch_size=32,
                                            class_mode='categorical',
                                            subset='validation')
```

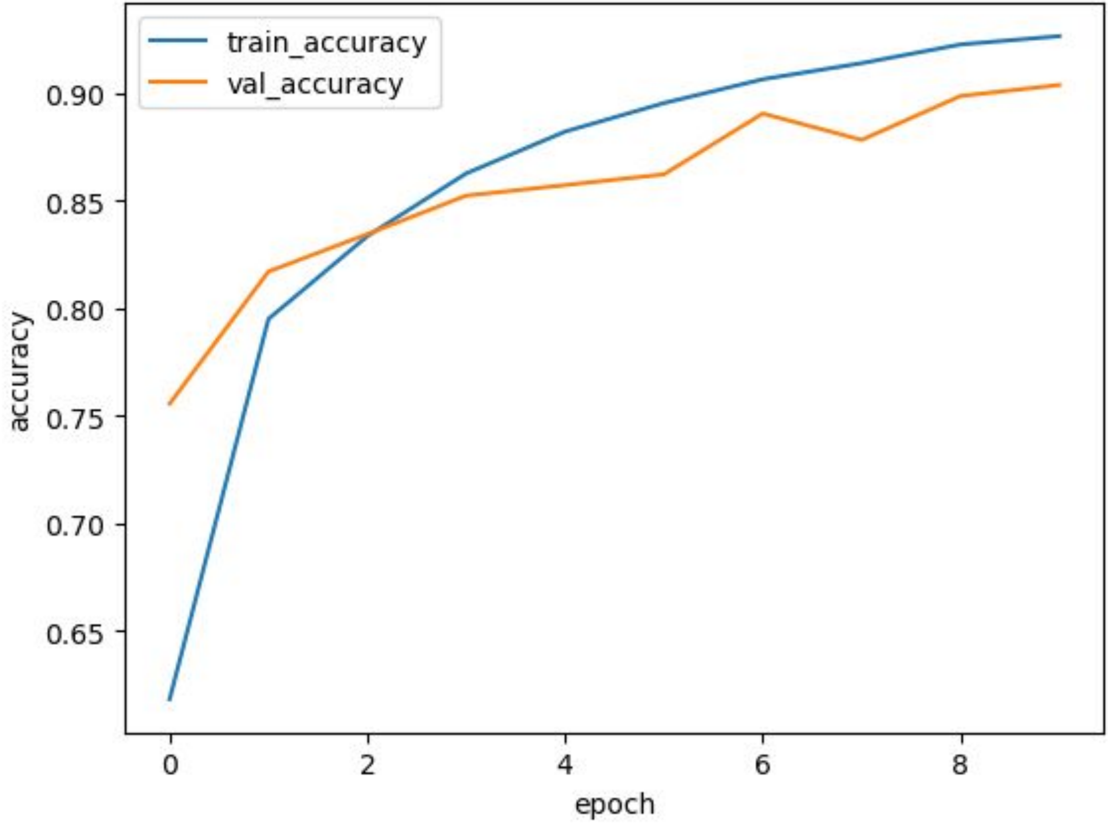
Building a Simple CNN Model

```
# Implement the CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(GlobalAveragePooling2D())
model.add(Dense(5, activation='softmax'))

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

# Train the model
history = model.fit(train_generator, validation_data=val_generator, epochs=10)
```

Result



Train

0.9276182

Validation

0.9243750

Pre-Trained Models

Keras

Search Keras documentation...

About Keras

Getting started

Developer guides

Keras 3 API documentation

Models API

Layers API

Callbacks API

Ops API

Optimizers

Metrics

Losses

Data loading

Built-in small datasets

Keras Applications

Xception

EfficientNet B0 to B7

EfficientNetV2 B0 to B3 and S, M, L

ConvNeXt Tiny, Small, Base, Large, XLarge

▶ Keras 3 API documentation / Keras Applications

Keras Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

Weights are downloaded automatically when instantiating a model. They are stored at `~/.keras/models/`.

Upon instantiation, the models will be built according to the image data format set in your Keras configuration file at `~/.keras/keras.json`. For instance, if you have set `image_data_format=channels_last`, then any model loaded from this repository will get built according to the data format convention "Height-Width-Depth".

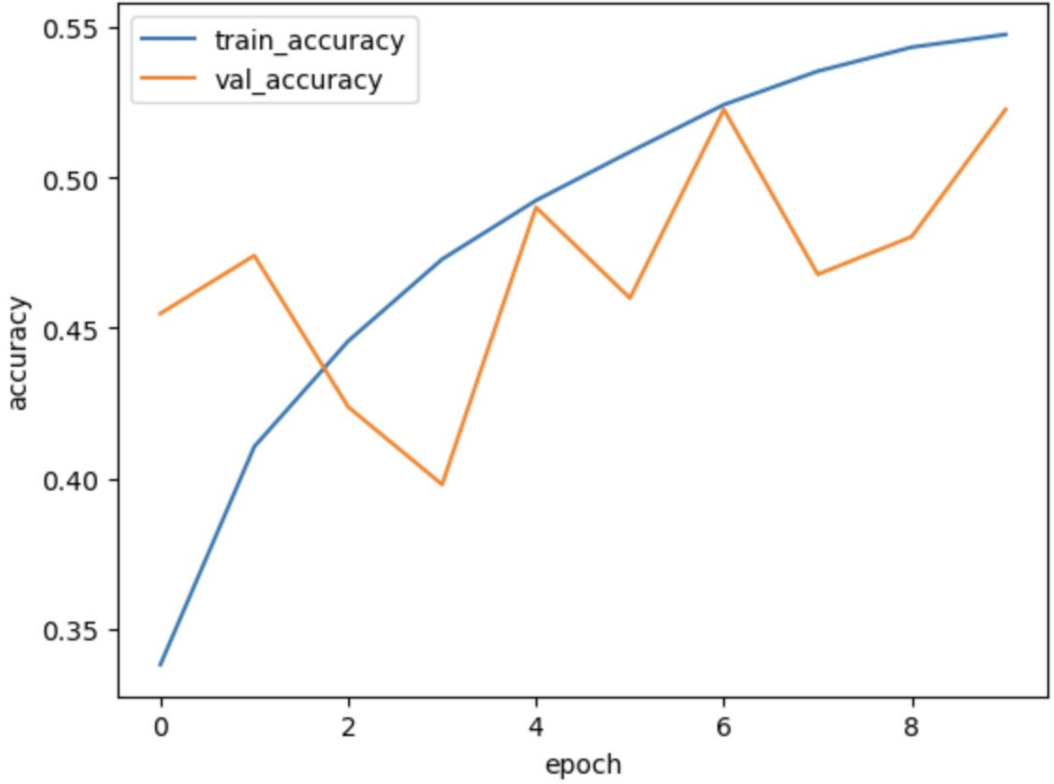
Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4

Keras Applications

- Available models
- Usage examples for image classification models
  - Classify ImageNet classes with ResNet50
  - Extract features with VGG16
  - Extract features from an arbitrary intermediate layer with VGG19
  - Fine-tune InceptionV3 on a new set of classes
  - Build InceptionV3 over a custom input tensor

Using Pre-trained Models- MobileNetV2



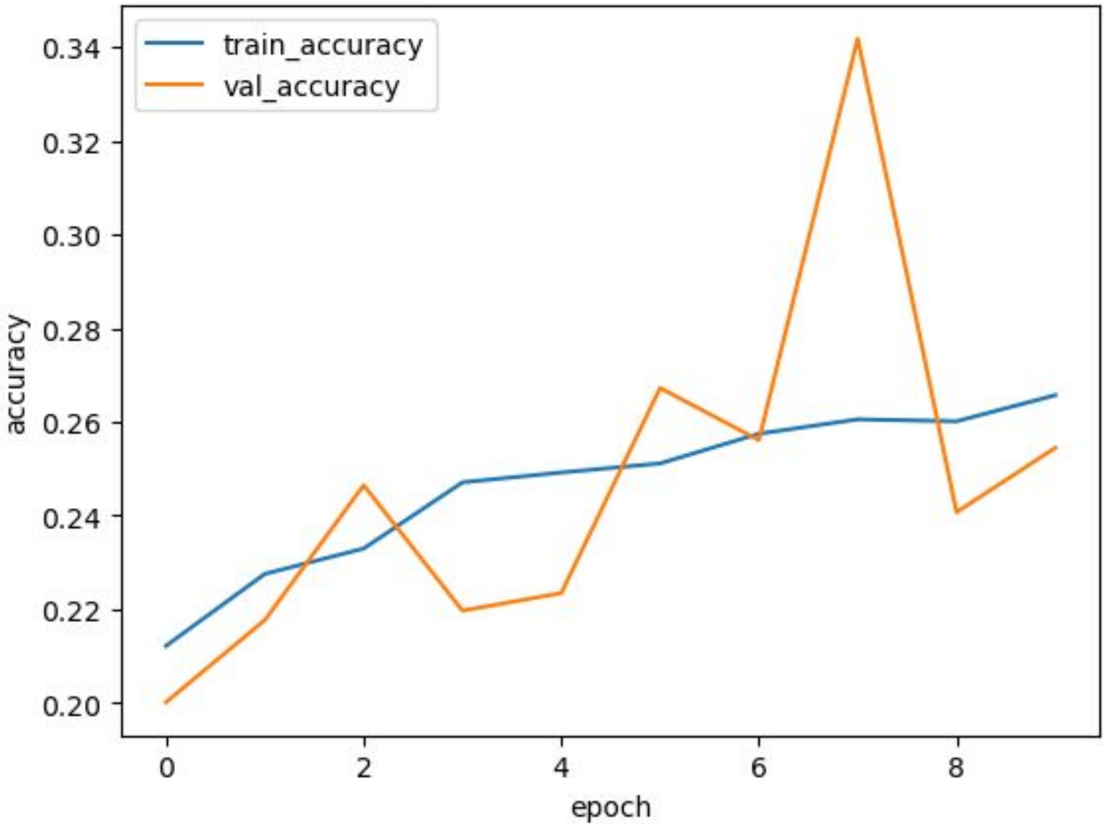
Train

0.5428

Validation

0.5226

Using Pre-trained Models- EfficientNetV2B0



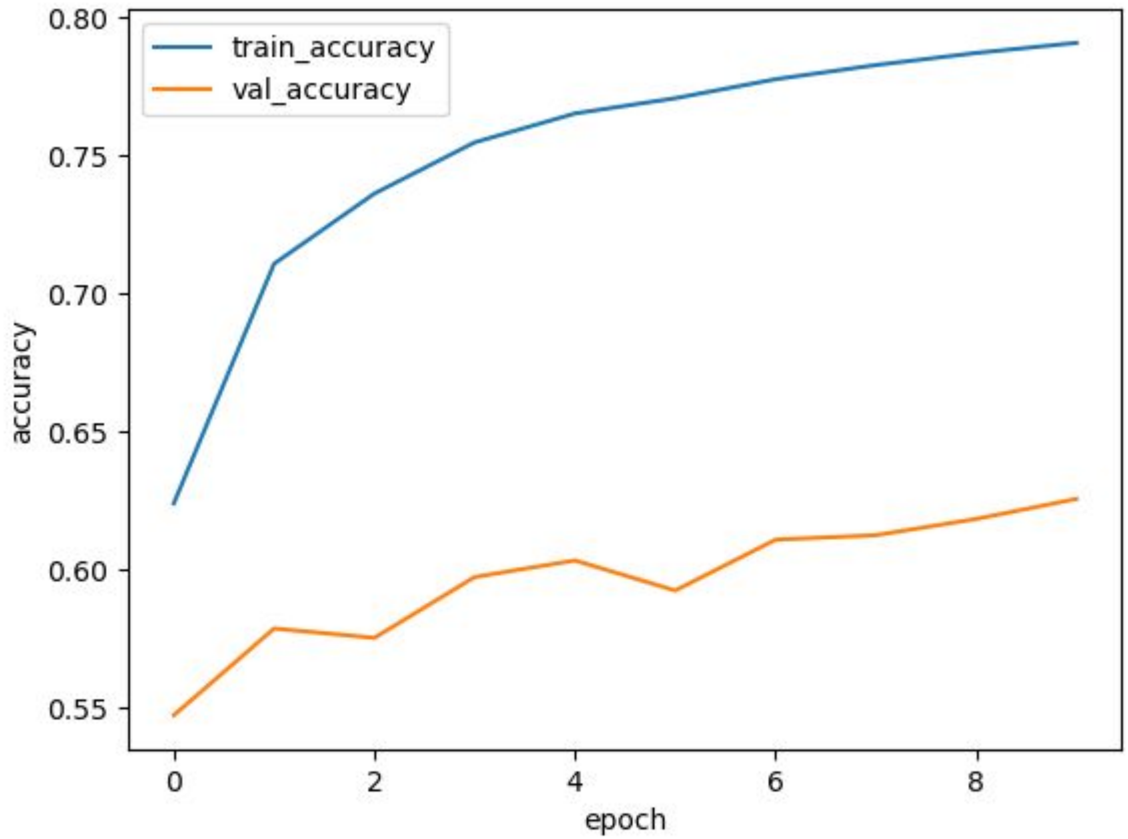
Train

0.2629

Validation

0.2545

Using Pre-trained Models- InceptionV3



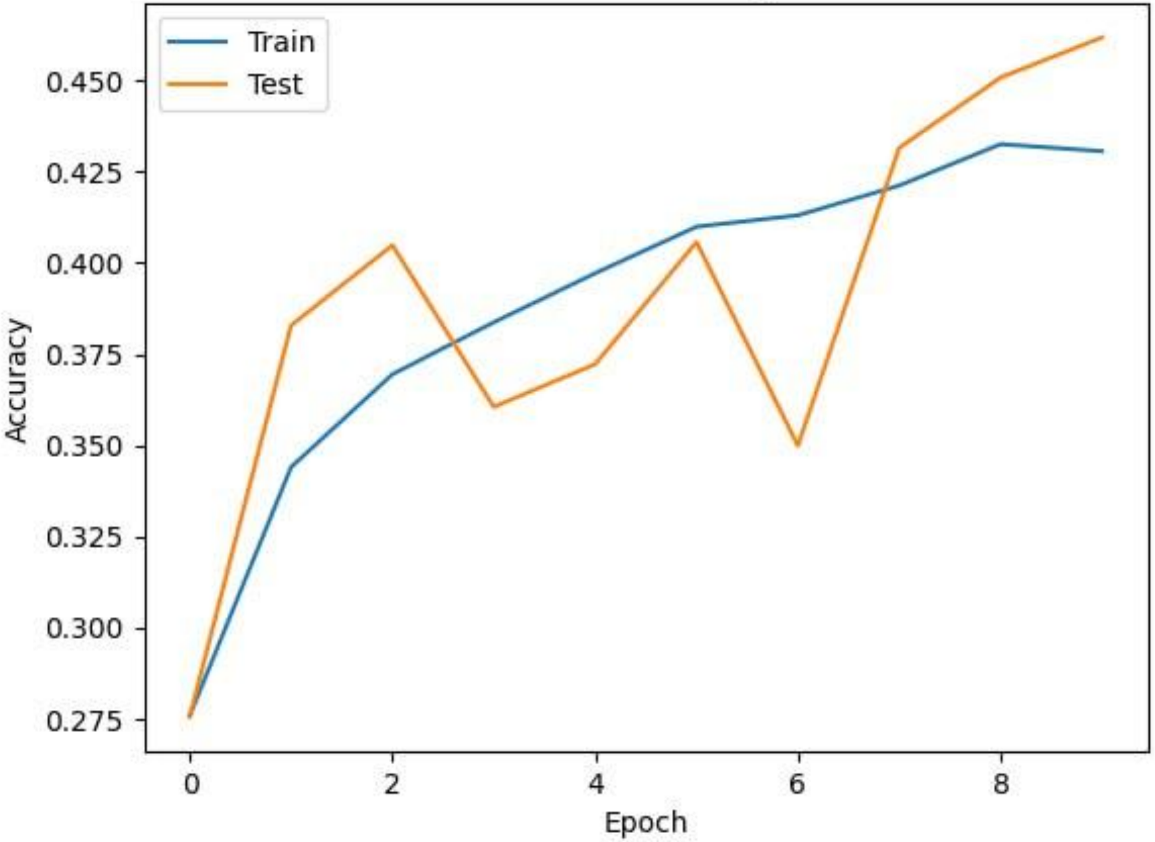
Train

0.7913

Validation

0.6257

Using Pre-trained Models- NASNetMobile



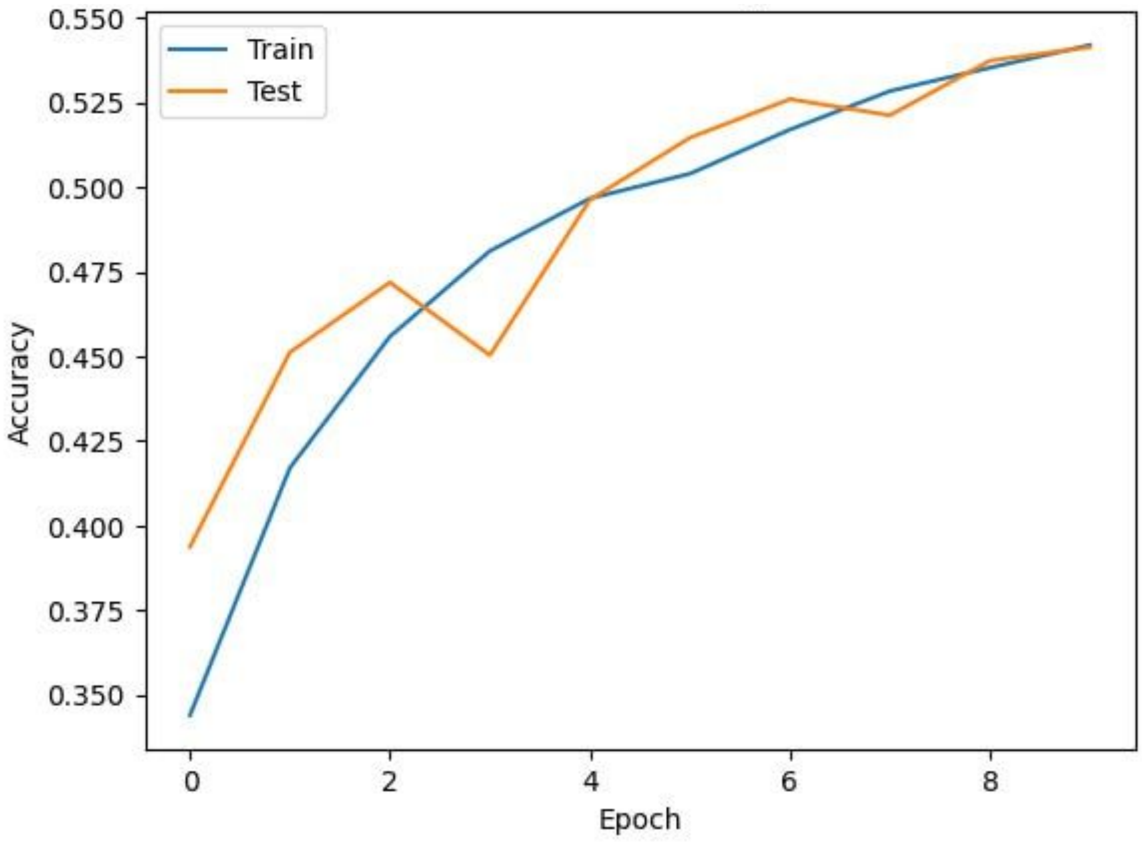
Train

0.4634

Validation

0.4651

Using Pre-trained Models- DenseNet121



Train

0.4643

Validation

0.4599

The Best Model

```
# Implement the CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(GlobalAveragePooling2D())
model.add(Dense(5, activation='softmax'))

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

# Train the model
history = model.fit(train_generator, validation_data=val_generator, epochs=10)
```

Evaluation

Possible Shortcomings

- Mismatch between pre-trained models' domain and Kaggle dataset
- Insufficient fine-tuning
- Overfitting or Underfitting
- Number of epochs

&

Areas for Future Research

- Exploring Other Model Architectures
- Fine-tuning
- Unsupervised and Semi-supervised Learning