Deep Neural Network-Based Canopy Image Classification

Research and Development Project

Mentored By
Prof. Gayathri Ananthanarayanan

Balsher Singh - 180010008

Karan Sharma - 180010019

Problem Statement for this project

In this work, the aim was to develop a deep neural network based canopy image classification framework for identifying different diseases and nutrient deficiencies in maize crop and also to identify and explore different optimizations to synergistically use the available compute hardware in the edge devices such that faster and reliable real time image inferencing can be done in the edge device itself rather than sending it to the cloud or a remote server.

Related Work

1. Classification of Macronutrient Deficiencies in Maize Plant Using Machine Learning

Journal: International Journal of Electrical and Computer Engineering (IJECE)

Dataset: Not available publicly. They used a set of only 100 images (75 for train & 25 for test)

Classification Classes:

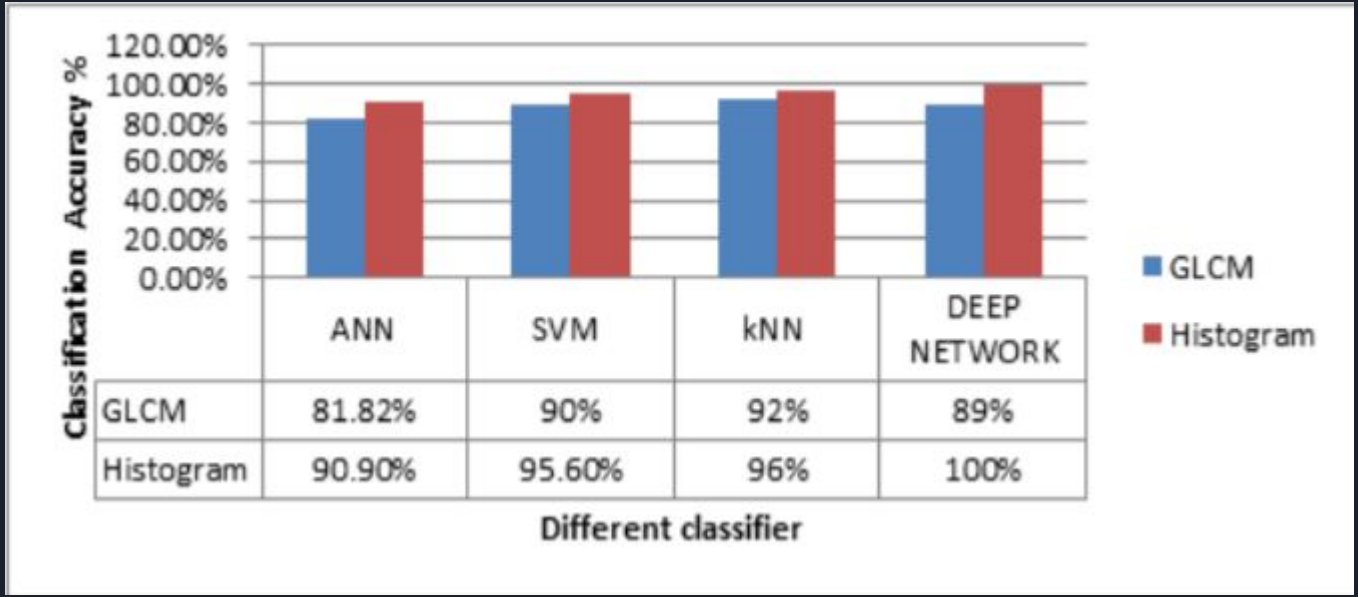
a) Nitrogen Deficient
b) Phosphorus Deficient
c) Potassium Deficient
d) Healthy

Methods Used:

a) ANN Classifier
b) SVM Classifier
c) kNN Classifier
d) Autoencoders
(No DNN used)

Handheld Device for validating: No

Result


Classification Accuracy %
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
ANN
SVM
kNN
DEEP NETWORK
GLCM
81.82%
90%
92%
89%
Histogram
90.90%
95.60%
96%
100%
GLCM
Histogram
Different classifier

2. Real-time detection of maize crop disease via a deep learning-based smartphone app

- Journal: SPIEDigitalLibrary.org/conference-proceedings-of-spie
- Dataset: Plant Village Dataset used for Healthy (1000 images) and Northern Leaf Blight (463 images).
- Classification Classes: Healthy leaf and Northern Leaf Blight
- Methods Used: Modified Deep CNN ResNet50 model (Input image size = 256x256x3)

Handheld Device for validating: Yes Android + iOS App.

Accuracy on Validation Dataset: 0.99

3. **Corzea: Portable Maize (Zea Mays L.) Nutrient Deficiency Identifier**

- **Journal and Issue Date:** INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 9, ISSUE **02, FEBRUARY 2020**

- **Dataset:** Training set - 5000 images, Test set - 200 images obtained from online resource www.image-net.org and from actual field (not publicly available)

- CNN Model
- **Classes:**
  - Phosphorus Deficient
  - Potassium Deficient
  - Nitrogen Deficient
  - Magnesium Deficient

Evaluation Metric:

$$Accuracy = \frac{(TN + TP)}{(TN + TP + FN + FP)} = \frac{NUMBER\ OF\ CORRECT\ ASSESSMENT}{NUMBER\ OF\ ALL\ ASSESSMENTS}$$

Result:

CLASSIFICATION RESULTS

Nutrient Deficient	Corn Leaf Samples	TP	TN	FP	FN	Accuracy
Phosphorus	50	47	0	6	0	94%
Potassium	50	47	0	10	0	94%
Nitrogen	50	42	0	12	0	84%
Magnesium	50	47	0	11	0	94%

4. Identification of Maize Leaf Diseases Using Improved Deep Convolutional Neural Networks

Dataset: 500 images from Plant Village Dataset and Google (Using Data Augmentation, 3060 images created for training and testing)

Classification Classes:

a) Southern leaf blight
b) Brown spot
c) Curvularia leaf spot
d) Rust
e) Dwarf mosaic
f) Gray leaf spot
g) Round spot
h) Northern leaf blight
i) Healthy

Methods Used: Modified Deep CNN GoogLeNet model (Input image size = 224x224x3)

Handheld Device for validating: No
Accuracy on Validation Dataset: 0.989

Maize Deficiency Detection

INTRODUCTION

- This is a binary classification with TensorFlow Model Garden Object Detection detecting any kind of nutrient deficiency in a maize plant leaf.

  Visit: [https://github.com/tensorflow/models]

DATASET USED

- Dataset provided by Prof. Gayathri. (Images clicked at UAS Dharwad)

  360 (260 train and 100 test)

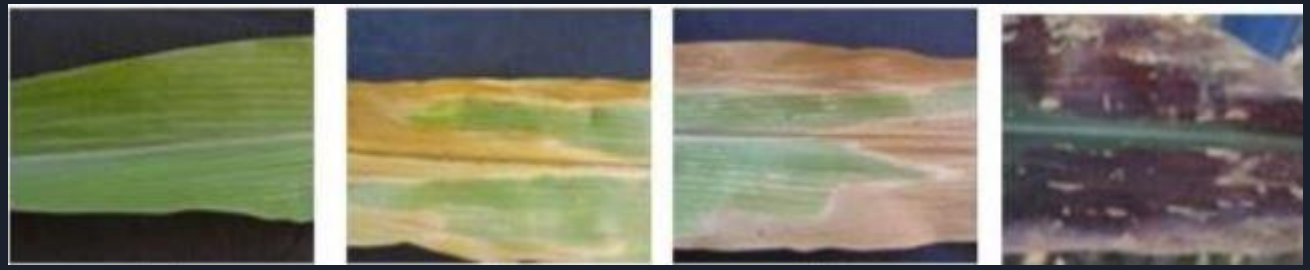


Fig: a.) Healthy, b.) Nitrogen Deficiency Leaf, c.) Phosphorous Deficiency ,d.) Potassium Deficiency

LABELING IMAGES IN DATASET

- Images labeled using “labelImg” [Visit: https://pypi.org/project/labelImg/]

TensorFlow Object detection Api used for training and detecting that a maize crop has nutrient deficiency or it’s a healthy plant.

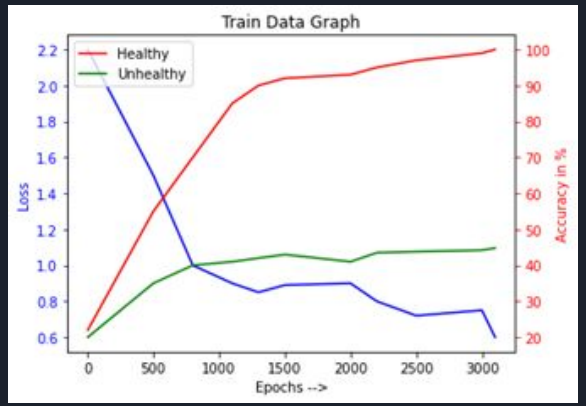
Since TensorFlow object Detection API predicts only Detection Box so by the following way the class of an image is predicted.

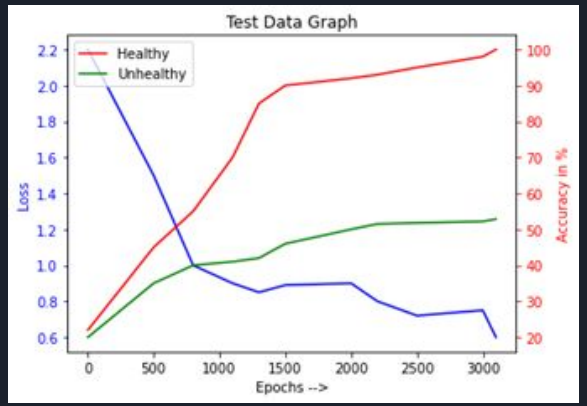
- 1 = ‘healthy’ and 2 = ‘unhealthy’
- weights_healthy = sum of detection score of all healthy box predicted
- weights_unhealthy = sum of all detection score of all unhealthy box predicted
- Class = roundoff ([(1 * weights_healthy) + (2 * weights_unhealthy)] /Total weights)

Models Trained :

- Efficientnet D3 (3100 steps, loss = 0.6)

Healthy Train	100%
Unhealthy Train	44.25%
Healthy Test	100%
Unhealthy Test	52.85%

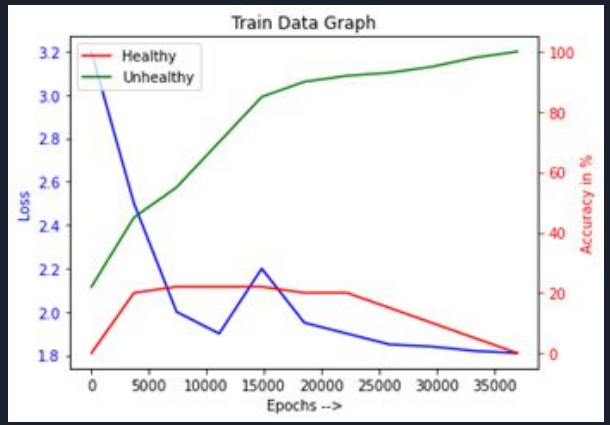


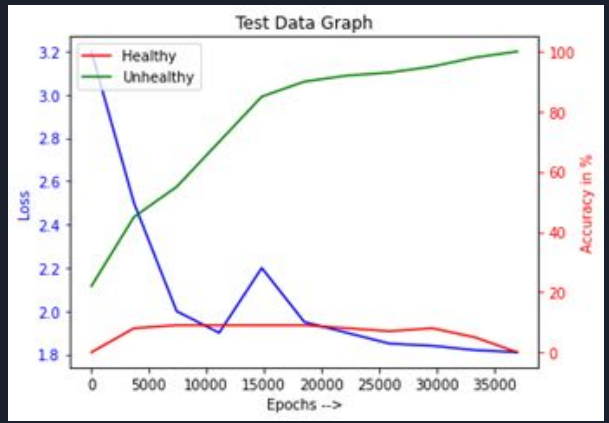


Models Trained :

- SSD Mobile net V2 320x320 (40000 steps, loss = 1.8)

Healthy Train	0%
Unhealthy Train	100%
Healthy Test	0%
Unhealthy Test	100%





Result for Maize Nutrient Deficiency Detection

Model 1 :

- Unhealthy is harder to train.
- Healthy accuracy is very high, maybe we should test more healthy images.

Model 2 :

- Overfitting on unhealthy images maybe because they were more in number

Models Trained :

- SSD MobileNet FPNLite (25000 steps, loss = 0.318)

Healthy Train	94.3%
Unhealthy Train	64.36%
Healthy Test	100%
Unhealthy Test	67%

- Observation
    - Unhealthy is harder to train
    - Healthy accuracy is very high, maybe we should test more healthy images

Models Trained :

- R-CNN inception (30900 steps, loss = 1.562)

Healthy Train	0%
Unhealthy Train	99.42%
Healthy Test	0%
Unhealthy Test	100%

- Observation
  - Very small dataset for such a large network
  - Overfitting on unhealthy images maybe because they were more in number

Limitation in Nutrient Deficient

Here, we had shortage of data due to which all models were suffering and not performing good at all.

Unavailability of large dataset

1. Only 260 images for training
2. Deep Learning works only with large dataset

Maize Disease Detection

Introduction

- Insufficient means of detecting/diagnosing diseases in crops is a big issue faced nationwide in fact worldwide, leading to scarce crop restoration procedures.
- In this application, we have worked on a solution that can be deployed in the field with ease and has the ability of recognizing diseases in maize (corn) leaf images, thus permitting pre-emptive corrective actions prior to significant yield loss.
- My approach is based on deep learning involving a convolutional neural network for both feature extraction and image classification to identify specific diseases in crops.
- It also involves the real-time implementation of such networks on android platforms for field deployment.

Datasets used ⟶

- Dataset provided by Prof. Gayathri. (Images clicked at UAS Dharwad)

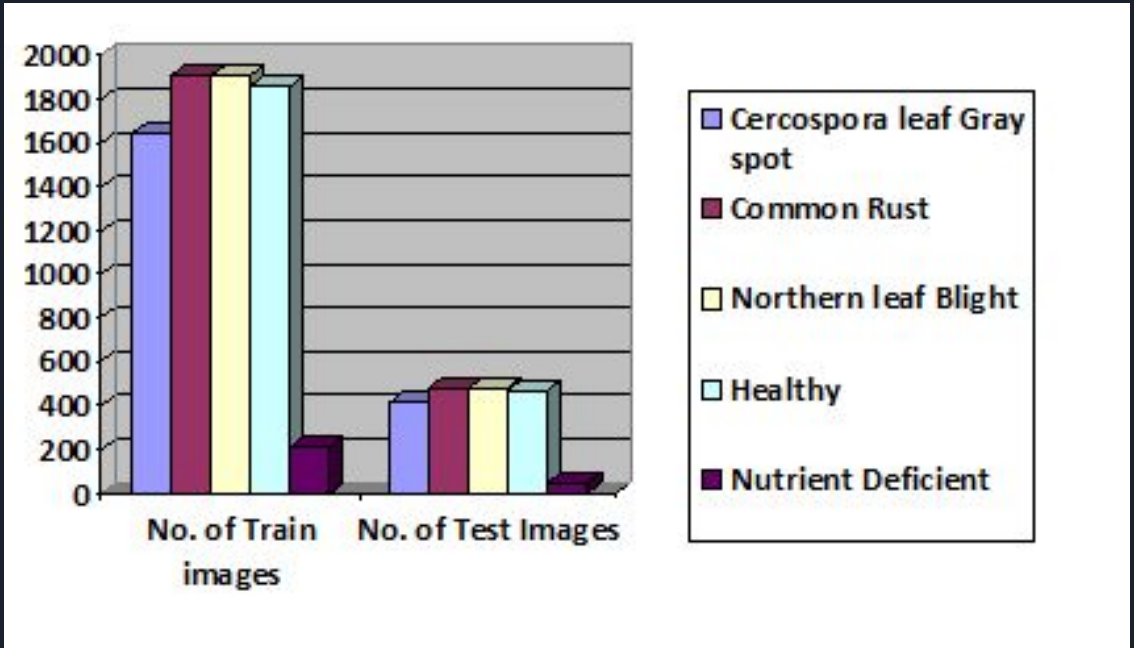
  360 (260 train and 100 test)

- Plant Village Dataset

  9509 (7580 train and 1929 test)

Fig: a.) Healthy, b.) Northern Leaf Blight, c.) Common Rust, d.) Cercospora Gray Leaf Spot e.) UnHealthy

Dataset


2000
1800
1600
1400
1200
1000
800
600
400
200
0
No. of Train images
No. of Test Images
Cercospora leaf Gray spot
Common Rust
Northern leaf Blight
Healthy
Nutrient Deficient

For Maize Disease Detection, we used **Modified Deep CNN VGG16** and **MobileNet V2** with **transfer learning** using **ImageNet Dataset weights** on these models.

Dataset Loaded with custom method ⟶

```
def create_dataset(img_folder):
  img_data_array=[]
  class_name=[]
  for dir1 in os.listdir(img_folder):
    print(dir1)
    for file in os.listdir(os.path.join(img_folder, dir1)):
      image_path= os.path.join(img_folder, dir1,  file)
      image= cv2.imread( image_path, cv2.COLOR_BGR2RGB)
      image=cv2.resize(image, (IMG_HEIGHT, IMG_WIDTH),interpolation = cv2.INTER_AREA
      image=np.array(image)
      image = image.astype('float32')
      image /= 255
      img_data_array.append(image)
      class_name.append(dir1)
  return img_data_array, class_name

NUM_CLASSES = 5
def convert_to_hot_encoded(t):
    y_data = []
    for index in range(len(t)):
        y_data.append(t[index])
    y_data_categorical = np_utils.to_categorical(y_data, NUM_CLASSES)
    return y_data_categorical
```

```
]  # Create x_train and y_train
   img_data, class_name =create_dataset(train_data_dir)
   target_dict={k: v for v, k in enumerate(np.unique(class_name))}
   target_val=  [target_dict[class_name[i]] for i in range(len(class_name))]
   x_train = np.array(img_data)
   y_train = convert_to_hot_encoded(np.array(target_val))

   Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot
   Corn_(maize)___Common_rust_
   Corn_(maize)___Northern_Leaf_Blight
   Corn_(maize)___healthy
   Corn_(maize)___unhealthy
```

- In the above methods, train data and test data is loaded from **'train_data_dir'** and **'test_data_dir'** into NumPy arrays **(x_train, y_train)** and **(x_test, y_test)** which can be used for training a CNN model.
- Input images are read from directory and resized to 224x224x3 and then added to above NumPy arrays using the custom function 'create_dataset'.

Models Trained

- MODIFIED VGG16

```
model_vgg16_conv = VGG16(input_shape=IMG_SHAPE, pooling='avg',weights='imagenet', include_top=False)
X = model_vgg16_conv.layers[-1].output
X = Dense(512, activation='relu', input_dim = (512,))(X)
X = Dropout(0.1)(X)
X = Dense(256, activation='relu')(X)
X = Dense(128, activation='relu')(X)
X = BatchNormalization()(X)
X = Dense(64, activation='relu')(X)
X = Dense(5, activation='softmax')(X)
my_model = Model(model_vgg16_conv.layers[0].output,X)
my_model.summary()
```

```
=================================================================
Total params: 15,150,661
Trainable params: 15,150,405
Non-trainable params: 256
```

- MODIFIED MobileNet V2

```
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,include_top=False,weights='imagenet')
X = base_model.layers[-1].output
X = tf.keras.layers.GlobalAveragePooling2D()(X)
X = Dense(5, activation='softmax')(X)
my_model = Model(base_model.layers[0].output,X)
my_model.summary()
```

```
==================================

Total params: 2,264,389

Trainable params: 2,230,277

Non-trainable params: 34,112
```

Optimizers Used

- ADAM
- SGD

Learning Rate

- **0.01**
- **0.1**

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.01)
adam = tf.keras.optimizers.Adam(learning_rate=0.01)
my_model.compile(optimizer=adam,
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

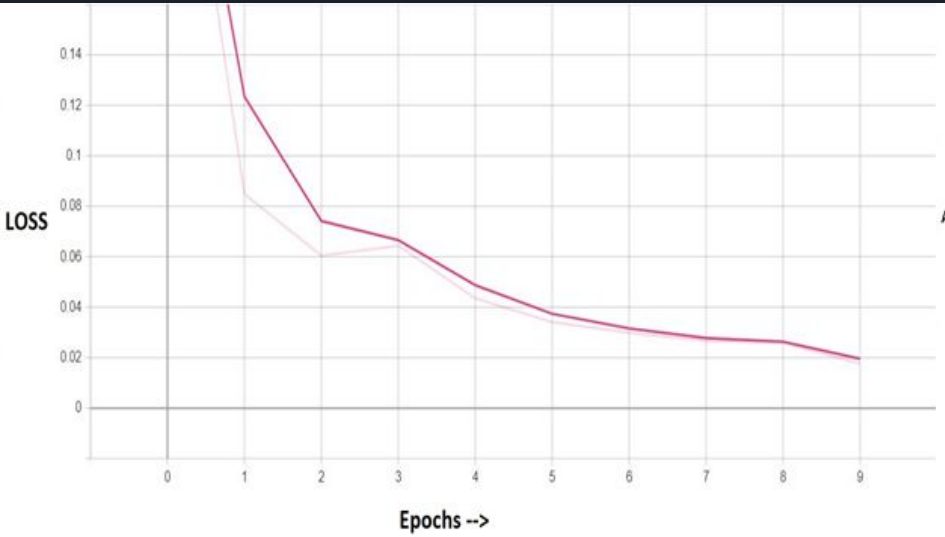
RESULT

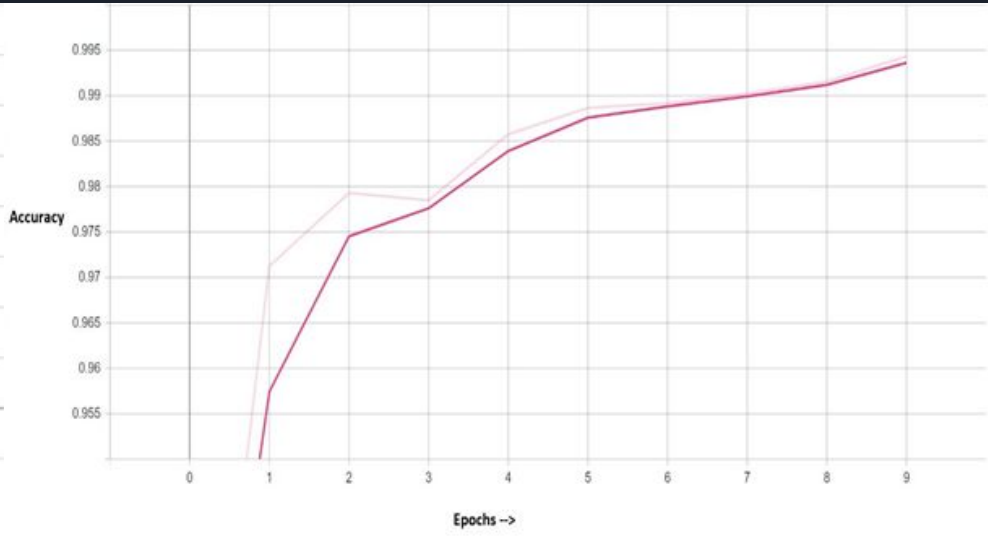
Model	Optimizer	Learning Rate	Train Accuracy	Test Accuracy
VGG16	SGD	0.01	99.8	98.76
		0.1	25.16	24.73
	Adam	0.01	25.71	25.66
		0.1	25.71	25.66
MobileNet V2	SGD	0.01	98.01	96.11
		0.1	97.8	73.20
	Adam	0.01	90.2	77.90
		0.1	25.71	25.66

- ** Train Accuracy is calculated from Tensorboard and also by importing saved model and then evaluating (x_train, y_train) on it.
- ** Test Accuracy is calculated by importing saved model and then evaluating (x_test, y_test) on it.

Modified VGG16

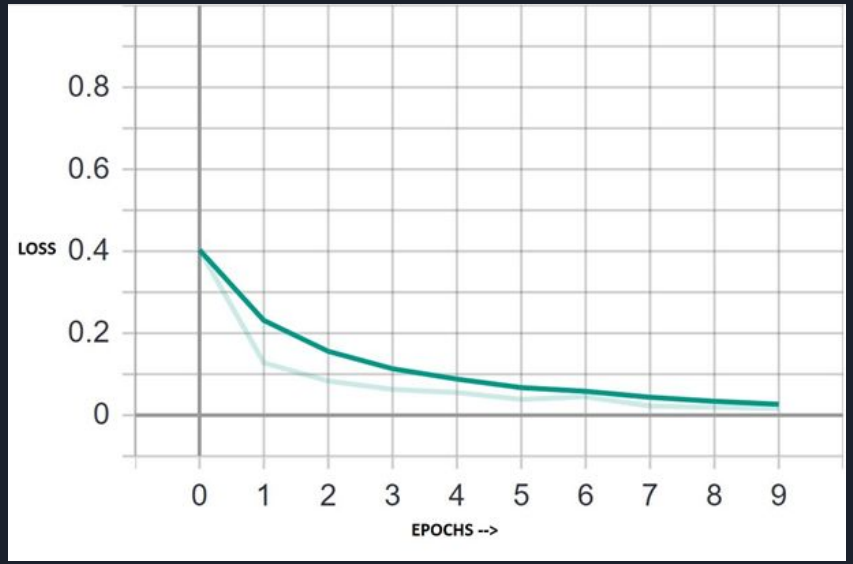
- [Optimizer: SGD & Learning Rate: 0.01]

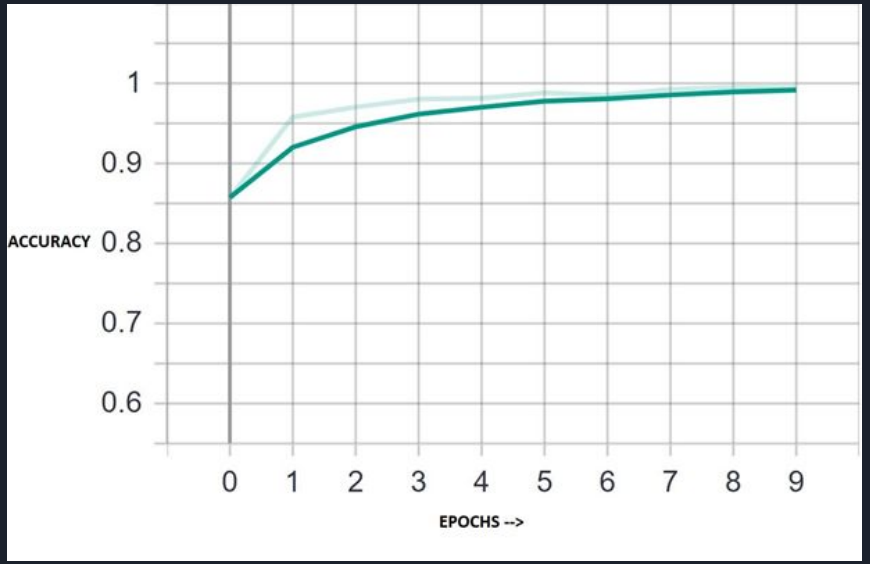




Modified MobileNet V2

- [Optimizer: SGD & Learning Rate: 0.01]



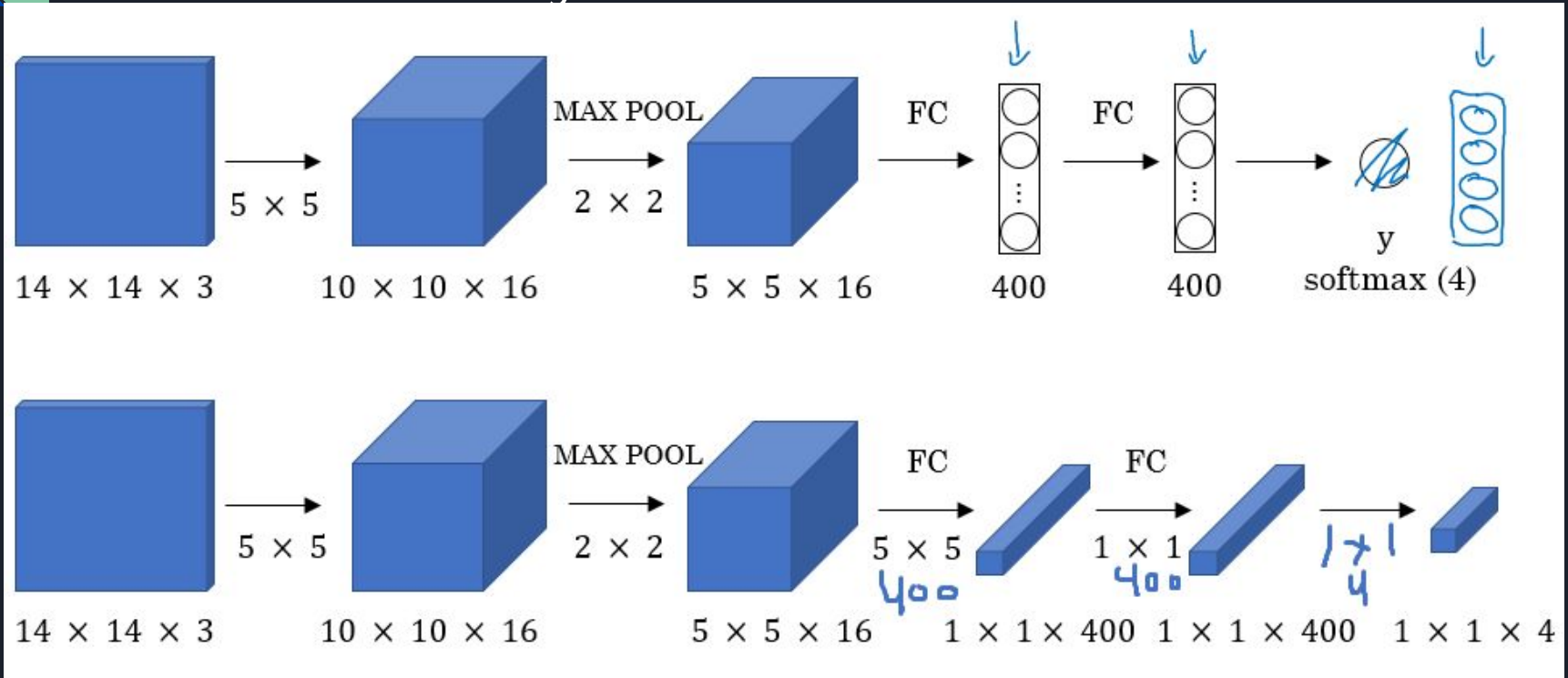


OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

- Integrated framework for Convolutional Networks for classification, localization and detection
- Solution for inefficiency caused by traditional sliding window
- Not a state of art anymore. So, why?
  - Nature of our dataset
  - We don’t need exact boundary box

Source: https://arxiv.org/abs/1312.6229

Turning Fully Connected layer to Convolution Layer

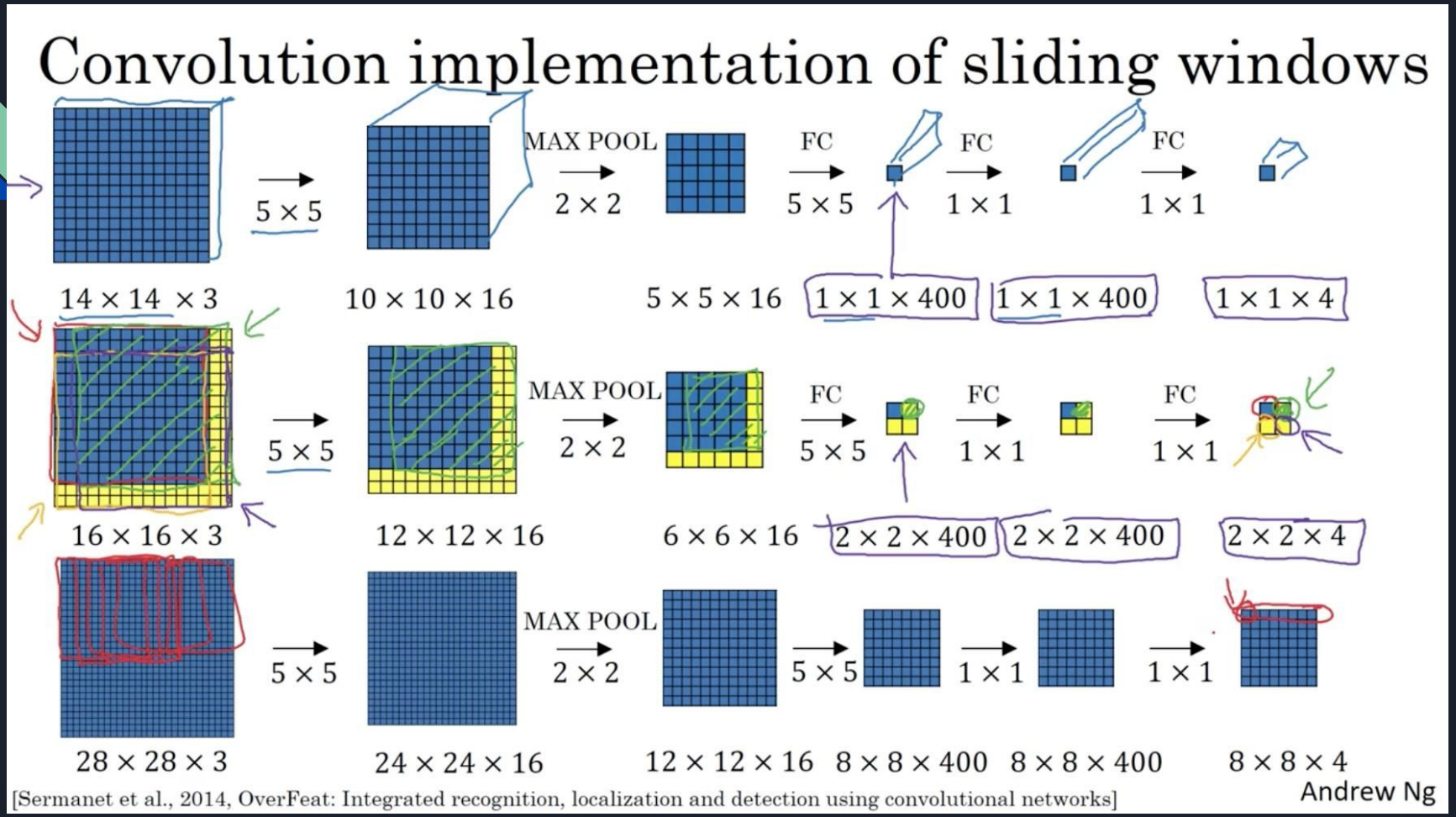


Credits: deeplearning.ai

Turning Fully Connected layer to Convolution Layer

```
base_model = tf.keras.applications.MobileNetV2(input_shape=(256,256,3),include_top=False,weights='imagenet')
model = tf.keras.Sequential([
                        base_model,
                        tf.keras.layers.BatchNormalization(),
                        tf.keras.layers.Conv2D(filters=5,kernel_size=8,activation='softmax'),
                        tf.keras.layers.Reshape((1,5)),
                        tf.keras.layers.Activation('softmax')
])
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 8, 8, 1280)        2257984
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 1280)        5120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 5)           409605
_________________________________________________________________
reshape_3 (Reshape)          (None, 1, 5)              0
_________________________________________________________________
activation (Activation)      (None, 1, 5)              0
=================================================================
Total params: 2,672,709
Trainable params: 2,636,037
Non-trainable params: 36,672
```



Credits: deeplearning.ai

Final model in OverFeat implementation

Modified model with different Input shape

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functi	(None, 20, 20, 1280)	2257984
batch_normalization (BatchNo	(None, 20, 20, 1280)	5120
conv2d (Conv2D)	(None, 13, 13, 5)	409605

Total params: 2,672,709
Trainable params: 2,636,037
Non-trainable params: 36,672

Methodology

- Loading Dataset

```
batch_size = 32
IMG_SIZE = (64,64)
```

```
data_dir = '/content/drive/My Drive/data/train'
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  seed=123,
  image_size=IMG_SIZE,
  batch_size=batch_size)
```

```
Found 7520 files belonging to 5 classes.
```

```
test_dir = '/content/drive/My Drive/data/valid'
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  test_dir,
  image_size=IMG_SIZE,
  batch_size=batch_size)
```

```
Found 1869 files belonging to 5 classes.
```

Methodology

- Normalization

```
normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```

```
0.08529412 1.0
```

- Sequential Model

```
base_model = tf.keras.applications.DenseNet121(input_shape=(64,64,3),include_top=False,weights='imagenet')
model = tf.keras.Sequential([
                             base_model,
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Conv2D(filters=5,kernel_size=2,activation='softmax'),
                             tf.keras.layers.Reshape((1,5)),
                             tf.keras.layers.Activation('softmax')
])
```

Methodology

- Decay Learning rate, loss function and compiling

```
initial_learning_rate = 0.01
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=50, decay_rate=0.96, staircase=False
)
```

```
def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits)
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=initial_learning_rate), loss=loss,metrics='accuracy')
```

- Callbacks during training
  - Saving checkpoint
  - early stopping
  - tensorboard events data

Methodology

```
checkpoint_dir = '/content/drive/My Drive/data/training7'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")
logdir = os.path.join("/content/drive/My Drive/logs7", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

early_stopping_cb = tf.keras.callbacks.EarlyStopping(
    patience=10, restore_best_weights=True
)

checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True,verbose=2,
    save_best_only=True)
```

- Model Training

```
history = model.fit(train_ds,epochs=30,
                    validation_data=val_ds,callbacks=[tensorboard_callback,checkpoint_callback,early_stopping_cb])
```

Result - Hyper Parameter set 1

- Base Model – MobileNet V2
- Model summary
- Input Image Size (256, 256)
- Normalization: True
- Decay learning rate
  - Initial learning rate = 0.01
  - Decay steps = 50
  - Decay rate = 0.96
- Optimizer – SGD

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 8, 8, 1280)        2257984
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 1280)        5120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 5)           409605
_________________________________________________________________
reshape_3 (Reshape)          (None, 1, 5)              0
_________________________________________________________________
activation (Activation)      (None, 1, 5)              0
=================================================================
Total params: 2,672,709
Trainable params: 2,636,037
Non-trainable params: 36,672
```

Result - Hyper Parameter set 1

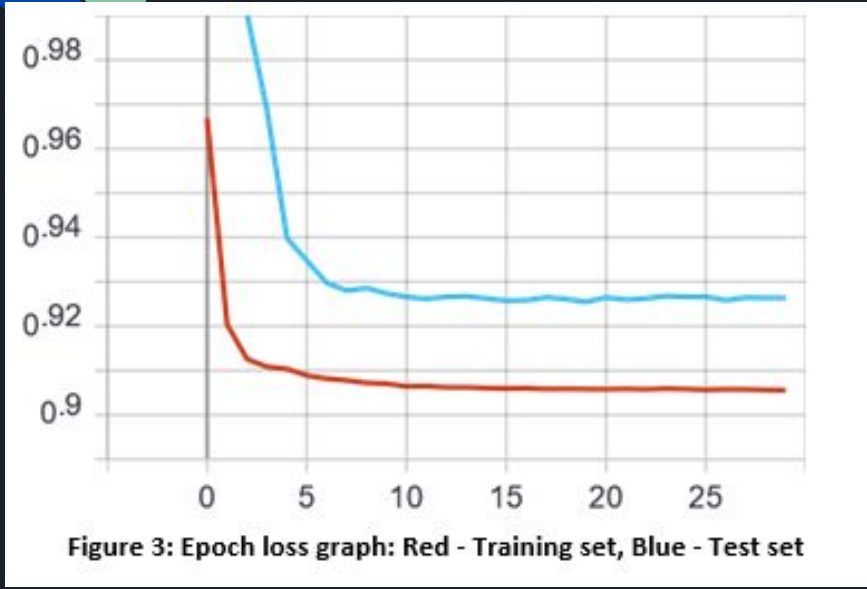


Figure 3: Epoch loss graph: Red - Training set, Blue - Test set

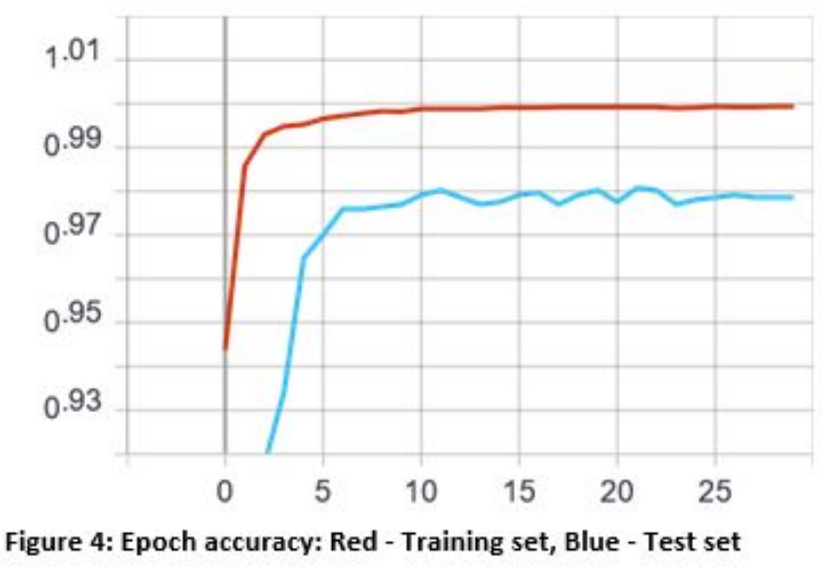


Figure 4: Epoch accuracy: Red - Training set, Blue - Test set

Best model is at 20th epoch with 99.92% accuracy on training set and 98.02% accuracy on test set.

Result - Hyper Parameter set 2

- Base Model – DenseNet121
- Input Image Size (64, 64)
- Model Summary
- Normalization: True
- Decay learning rate
  - Initial learning rate = 0.01
  - Decay steps = 50
  - Decay rate = 0.96
- Optimizer – SGD

Layer (type)	Output Shape	Param #
densenet121 (Functional)	(None, 2, 2, 1024)	7037504
batch_normalization (BatchNo	(None, 2, 2, 1024)	4096
conv2d (Conv2D)	(None, 1, 1, 5)	20485
reshape (Reshape)	(None, 1, 5)	0
activation (Activation)	(None, 1, 5)	0

Total params: 7,062,085
Trainable params: 6,976,389
Non-trainable params: 85,696

Result - Hyper Parameter set 2

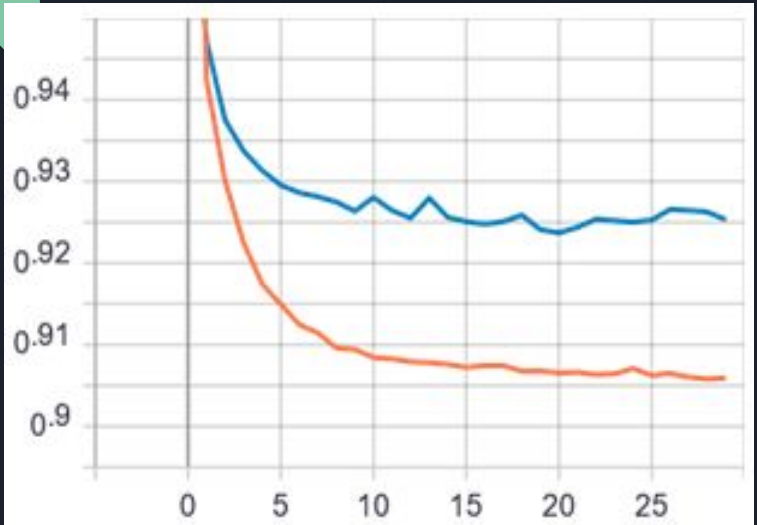


Figure 5 Epoch loss graph: Orange-Train, Blue- Test

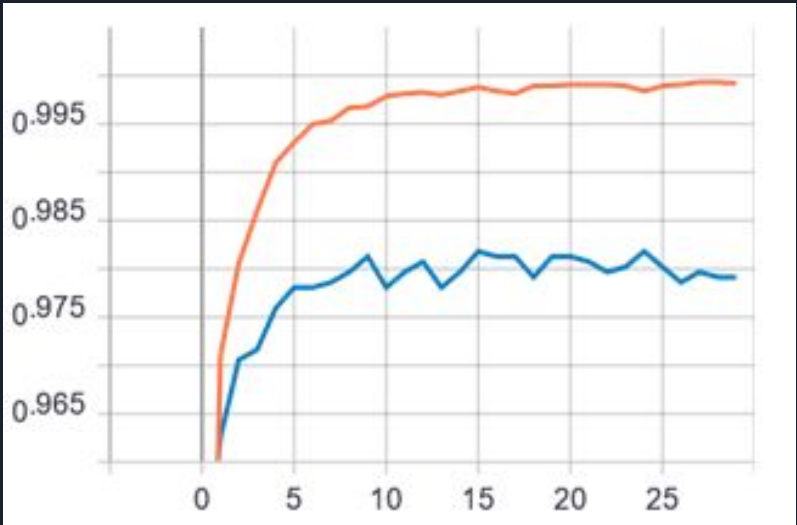


Figure 6 Epoch accuracy graph: Orange-Train, Blue- Test

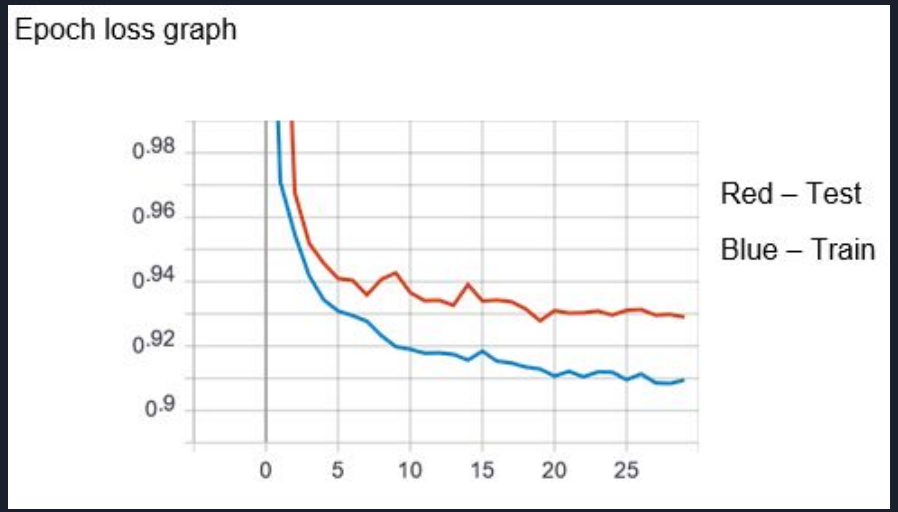
Best model is at 20th epoch with 99.89% accuracy on training set and 98.13% accuracy on test set.

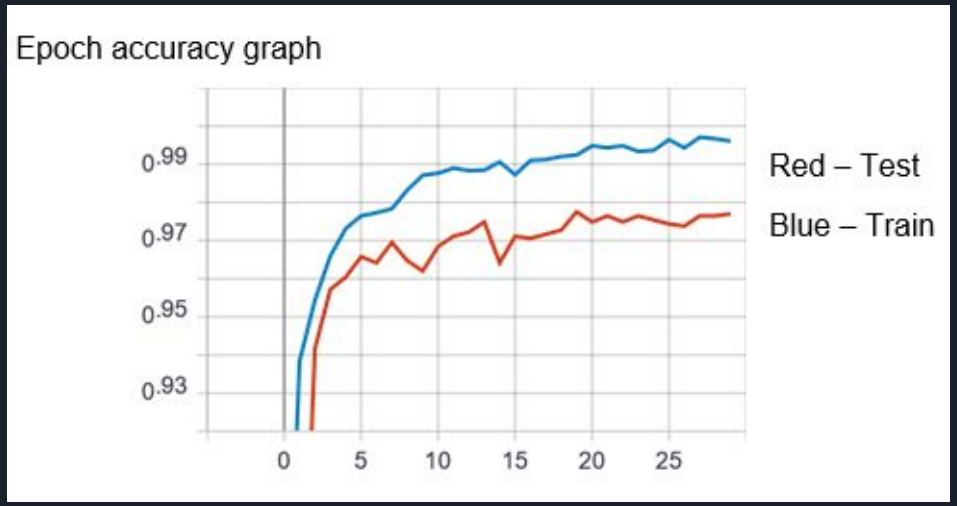
Result - Hyper Parameter set 3

- Base Model – MobileNet V2
- Model Summary
- Input Image Size (64,64)
- Normalization: True
- Decay learning rate
- Initial learning rate = 0.01
  - Decay steps = 50
  - Decay rate = 0.96
- Optimizer – SGD

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 2, 2, 1280)        2257984
_________________________________________________________________
batch_normalization (BatchNo (None, 2, 2, 1280)        5120
_________________________________________________________________
conv2d (Conv2D)              (None, 1, 1, 5)           25605
_________________________________________________________________
reshape (Reshape)            (None, 1, 5)              0
_________________________________________________________________
activation (Activation)      (None, 1, 5)              0
=================================================================
Total params: 2,288,709
Trainable params: 2,252,037
Non-trainable params: 36,672
_________________________________________________________________
```

Result - Hyper Parameter set 3





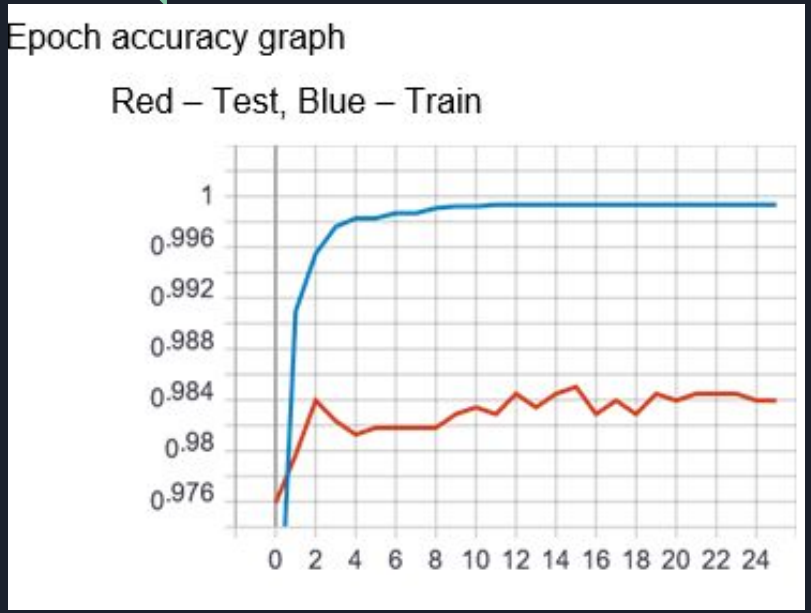
Best test accuracy of model is at 20th epoch with 99.24% accuracy on training set and 97.75% accuracy on test set.

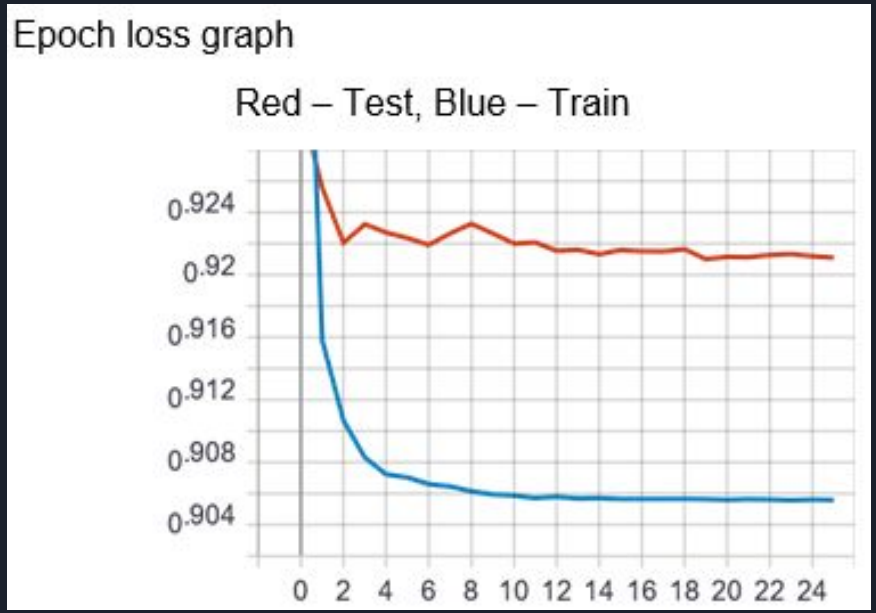
Result - Hyper Parameter set 4

- Base Model – DenseNet121
- Input Image Size (256, 256)
- Model Summary
- Normalization: True
- Decay learning rate
  - Initial learning rate = 0.001
  - Decay steps = 50
  - Decay rate = 0.96
- Optimizer - SGD

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Functional)     (None, 8, 8, 1024)        7037504
_________________________________________________________________
batch_normalization (BatchNo (None, 8, 8, 1024)        4096
_________________________________________________________________
conv2d (Conv2D)              (None, 1, 1, 5)           327685
_________________________________________________________________
reshape (Reshape)            (None, 1, 5)              0
_________________________________________________________________
activation (Activation)      (None, 1, 5)              0
=================================================================
Total params: 7,369,285
Trainable params: 7,283,589
Non-trainable params: 85,696
```

Result - Hyper Parameter set 4





Best test accuracy of model is at 20th epoch with 99.93% accuracy on training set and 98.45% accuracy on test set.

Result - Hyper Parameter set 5

- Base model – MobileNet V2
- Input Image Size (256, 256)
- Normalization: True
- Decay learning rate
  - Initial learning rate = 0.001
  - Decay steps = 50
  - Decay rate = 0.96
- Optimizer - Adam
- **Train accuracy – 74.84%**
- **Test accuracy- 75.49%**

Result - Hyper Parameter set 5

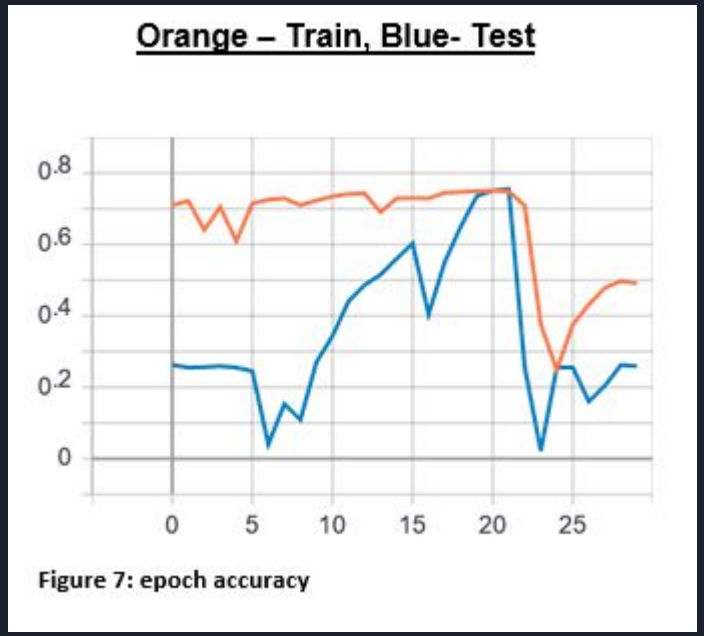


Figure 7: epoch accuracy

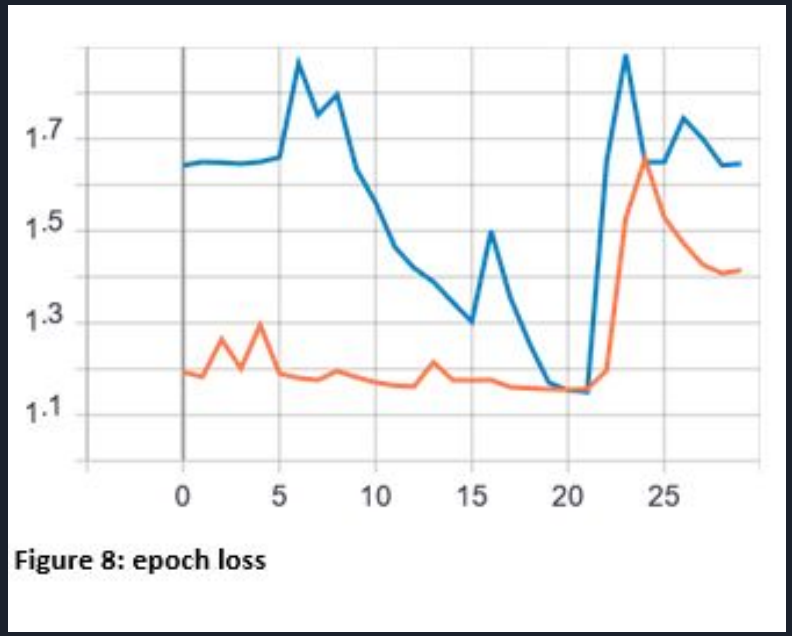


Figure 8: epoch loss

Result - Hyper Parameter set 6

- Base model – MobileNet V2
- Input Image Size (256, 256)
- Normalization: True
- Fixed learning rate = 0.001
- Optimizer - Adam
- **Train accuracy – 70.24%**
- **Test accuracy- 71.80%**

Result - Hyper Parameter set 6

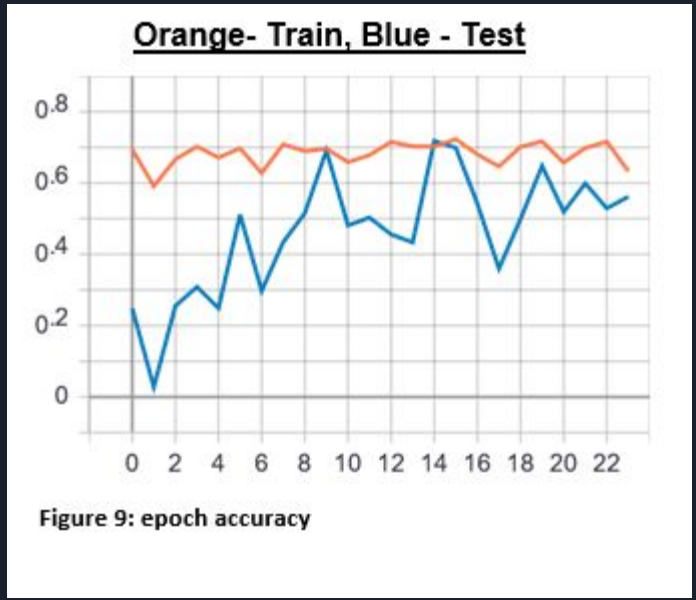


Figure 9: epoch accuracy

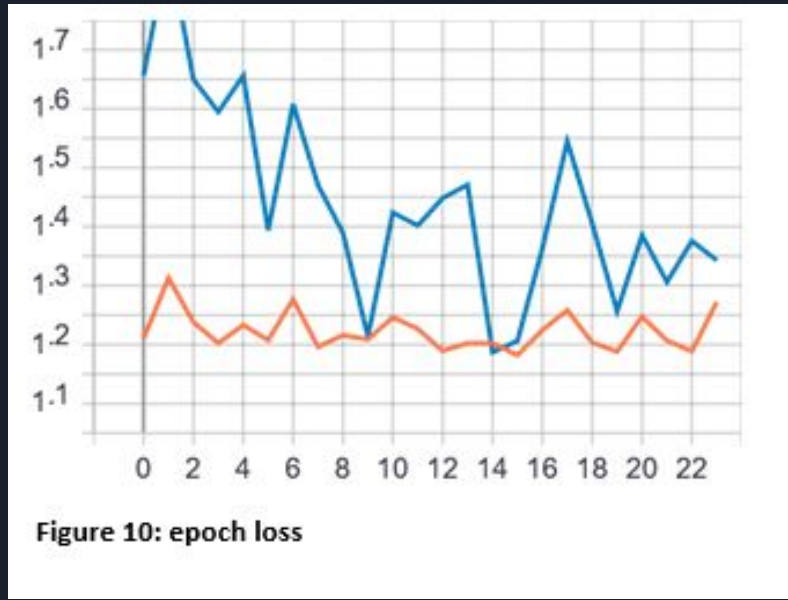


Figure 10: epoch loss

General Observation

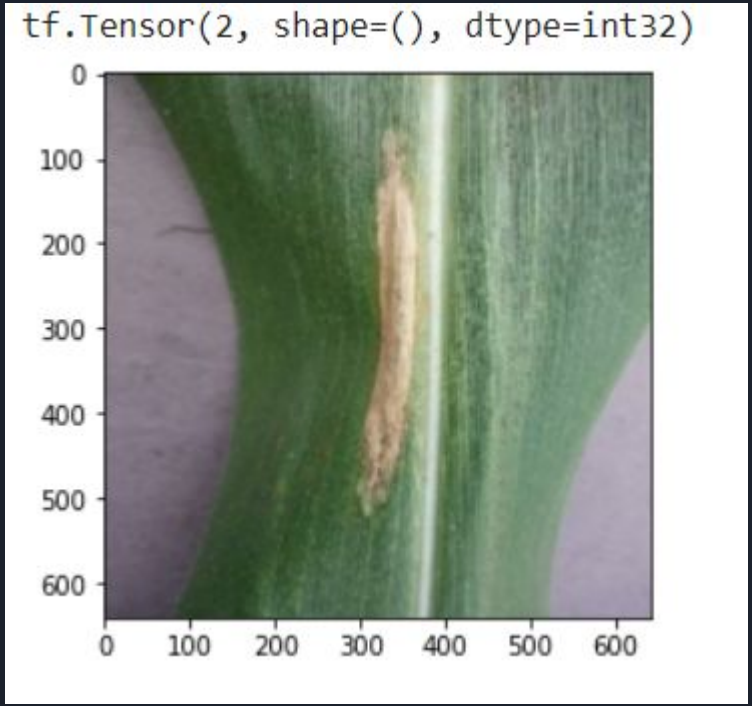
- SGD optimizers perform much better than Adam
- Adam optimizer performance is unsatisfactory.
- Decay learning rate is better than fixed learning rate
- DenseNet which has more depth perform better than MobileNet model
- Input size (256,256) perform better than (64,64)

Candidate Model

	MobileNetv2	DenseNet121
Input Size	(256,256,3)	(256,256,3)
No. of parameters	2.2 M	7.3 M
Depth	88	121
Test Accuracy	98.02%	98.45%
Hyperparameter	Set1	Set4

Final model in OverFeat implementation

Input (640,640,3)



Output

```
[[3. 3. 3. 3. 3. 3. 3. 2. 3. 3. 3. 3. 0.]
 [3. 3. 3. 3. 3. 3. 2. 2. 2. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 2. 2. 2. 3. 3. 3. 0.]
 [3. 3. 3. 3. 3. 3. 2. 2. 2. 2. 3. 3. 0.]
 [3. 3. 3. 3. 3. 3. 3. 2. 2. 3. 3. 3. 3.]
 [3. 3. 3. 0. 0. 3. 3. 2. 2. 3. 3. 3. 0.]
 [3. 3. 4. 3. 0. 0. 2. 2. 2. 3. 0. 3. 3.]
 [3. 3. 4. 3. 2. 2. 2. 2. 2. 0. 0. 3. 3.]
 [3. 3. 3. 3. 2. 2. 2. 2. 2. 0. 2. 3. 3.]
 [3. 3. 3. 3. 2. 2. 2. 2. 2. 0. 0. 3. 3.]
 [3. 3. 3. 3. 0. 2. 2. 2. 2. 2. 2. 3. 3.]
 [3. 3. 0. 0. 3. 2. 2. 2. 2. 0. 3. 3. 3.]
 [3. 3. 3. 0. 0. 0. 0. 2. 0. 0. 3. 3. 3.]]
```

Deployment in Android

The best Model is VGG16 [SGD,0.01] whose Test Accuracy is 98.76 %.

And 2nd best MobileNetV2 [SGD,0.01] whose Test Accuracy is 96.11%

MobileNetV2 has just 2,264,389 Parameters as compared VGG16 which has 15,150,661 Parameters.

** Best for Mobile App: MobileNetV2 [SGD,0.01]

- In order to use Trained model in Android App, we need to convert TensorFlow Model into .tflite Model

```
[ ]  # Converting a SavedModel to a TensorFlow Lite model.
     import tensorflow as tf
     saved_model_dir = '/content/gdrive/My Drive/tensorflow/data/saved_model/MN_SGD_0point01/'
     # Convert the model
     converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir) # path to the SavedModel directory
     tflite_model = converter.convert()
     # Save the model.
     with open('MN.tflite', 'wb') as f:
       f.write(tflite_model)
```

Dependencies ⟶

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

    //Custom Here
    implementation('org.tensorflow:tensorflow-lite:0.0.0-nightly') { changing = true }
    implementation('org.tensorflow:tensorflow-lite-gpu:0.0.0-nightly') { changing = true }
    implementation('org.tensorflow:tensorflow-lite-support:0.0.0-nightly') { changing = true }

    implementation 'com.camerakit:camerakit:1.0.0-beta3.11'
    implementation 'com.camerakit:jpegkit:0.1.0'
    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.0'
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.0.0'

}
```

In order to use camera in our app, we used an open source library ‘CameraKit’.

Link: https://github.com/CameraKit/camerakit-android

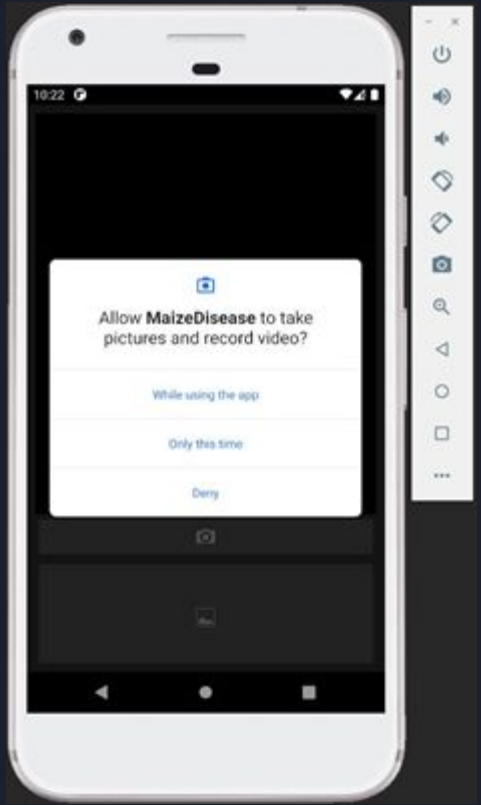
Website: https://camerakit.io/

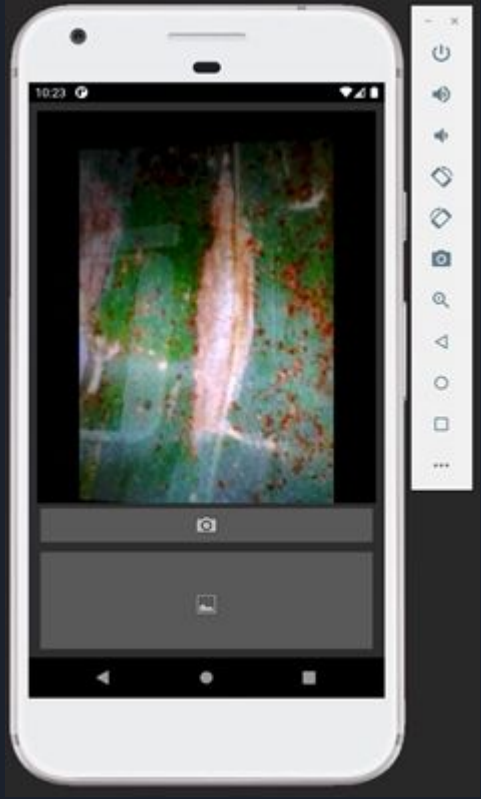
```
imageButton.setOnClickListener((v) → {
        cameraKitView.captureImage((cameraKitView, capturedImage) → {
                Bitmap bitmap = BitmapFactory.decodeByteArray(capturedImage,  offset: 0, capturedImage.length);
                Bitmap resized = Bitmap.createScaledBitmap(bitmap,  dstWidth: 224,  dstHeight: 224,  filter: true);
                Global.setCapturedImage(capturedImage);
                Global.setBitmap(resized);
                //MediaStore.Images.Media.insertImage(getContentResolver(), resized, "test.jpg" ,"New");
                startActivity(new Intent( packageContext: MainActivity.this, ImageClickedActivity.class));
        });
});
```

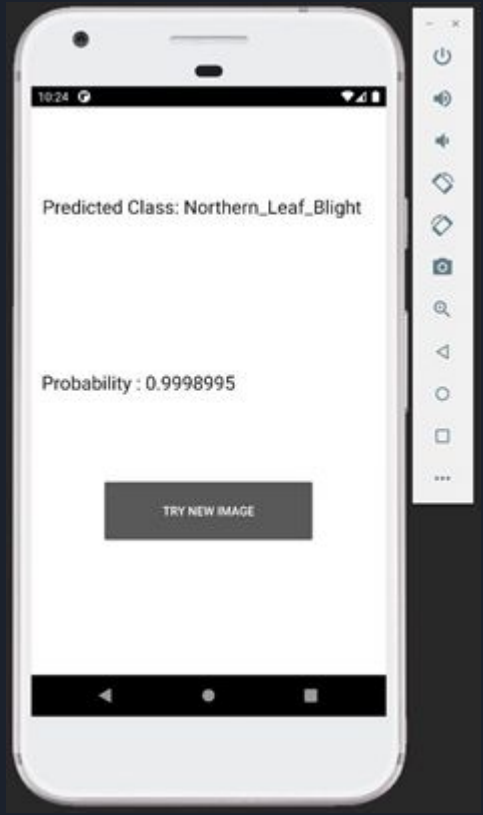
In order to use TensorFlow model, we used **org.tensorflow:tensorflow-lite:0.0.0-nightly library**.

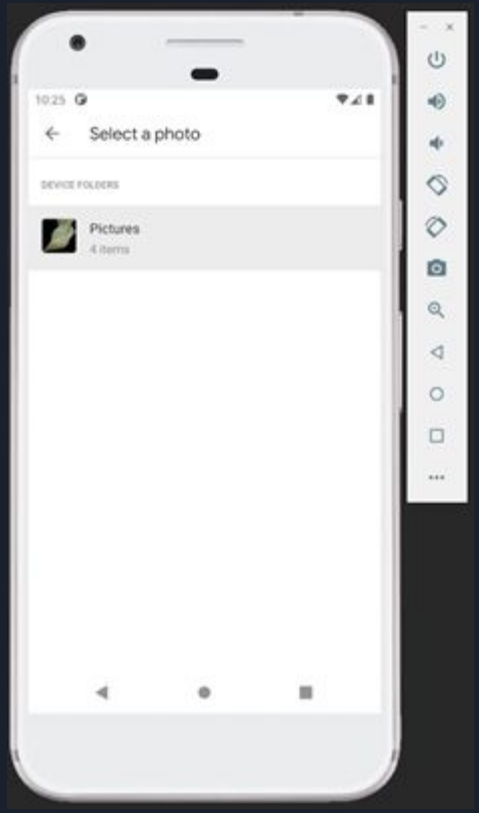
```
try {
    tfliteModel = FileUtil.loadMappedFile( context: ImageClickedActivity.this, modelFile);
    labels = FileUtil.loadLabels( context: ImageClickedActivity.this,label);
    tflite = new Interpreter(tfliteModel);
} catch (IOException e) {
    e.printStackTrace();
}
ByteBuffer byteBuffer = convertBitmapToByteBuffer(Global.getBitmap());
ByteBuffer outputBuffer = ByteBuffer.allocateDirect(4*5);
outputBuffer.order(ByteOrder.nativeOrder());
float[][] out = new float[1][5];
Object[] input = {byteBuffer};
Map<Integer, Object> outputs = new HashMap();
outputs.put(0, out);
tflite.runForMultipleInputsOutputs(input, outputs);
```

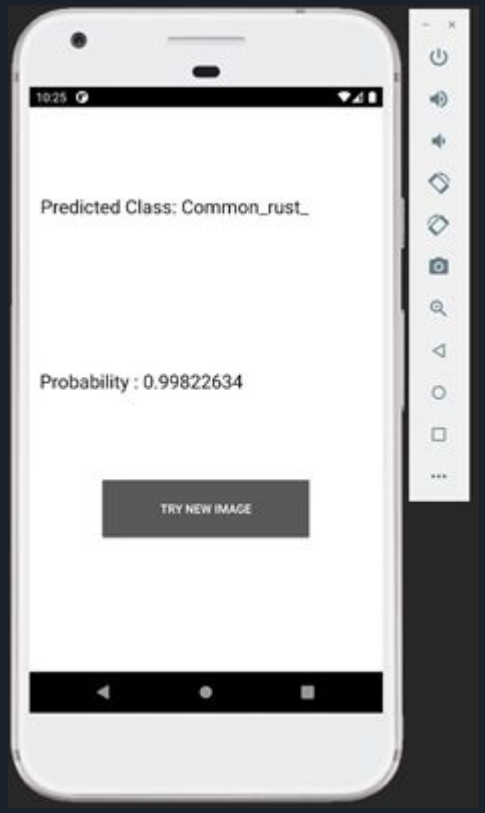
Some Screenshots from the app:








10:25
Select a photo
DEVICE FOLDERS
Pictures
4 items


10:25
Predicted Class: Common_rust_
Probability : 0.99822634
TRY NEW IMAGE

Time taken

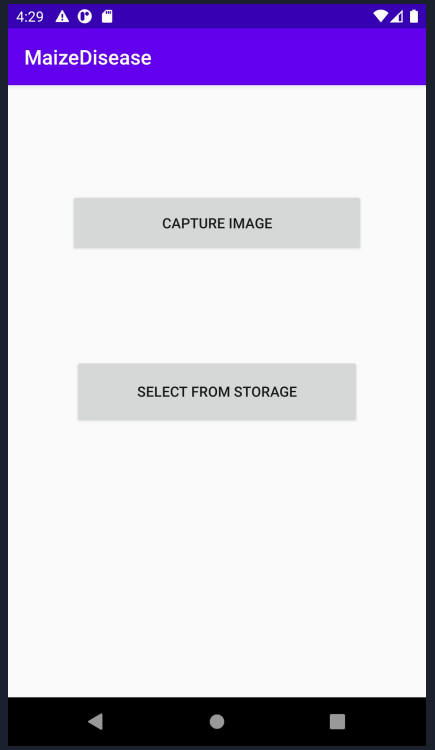
```
E/CameraException: Flash: FLASH_AUTO is not supported in this version of CameraKit.
I/CameraManagerGlobal: Connecting to camera service
W/Gralloc4: allocator 3.x is not supported
W/le.maizediseas: [SurfaceTexture-1-30579-0] bindTextureImage: clearing GL error: 0x506
I/tflite: Initialized TensorFlow Lite runtime.
I/System.out: Time taken is : 121
I/System.out: [[0.38069108, 0.5560788, 0.01831147, 0.034575313, 0.010343308]]
I/System.out: 0.5560788
    Common_rust_
E/CameraException: Flash: FLASH_AUTO is not supported in this version of CameraKit.
I/System.out: Time taken is : 116
    [[0.0067165573, 0.991814, 2.6594407E-8, 0.0014694561, 4.4564953E-8]]
    0.991814
I/System.out: Common_rust_
E/CameraException: Flash: FLASH_AUTO is not supported in this version of CameraKit.
I/System.out: Time taken is : 113
I/System.out: [[0.17339978, 0.14660782, 8.1591206E-5, 0.6798944, 1.6378313E-5]]
    0.6798944
    Northern_Leaf_Blight
```

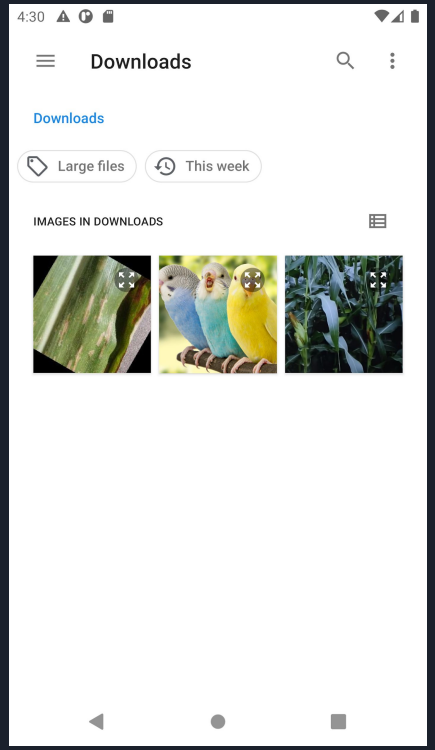
Deployment in Android

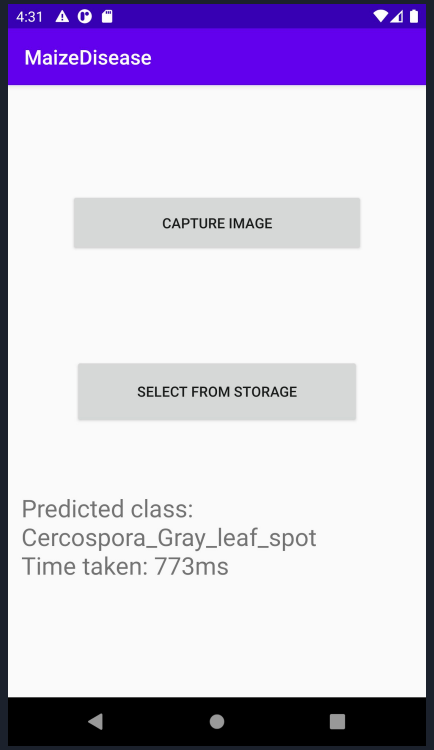
- Model trained with Hyperparameter set 1 is used in Android app

```
try {
    MappedByteBuffer tfliteModel
            = FileUtil.loadMappedFile( context: MainActivity.this,
            filePath: "model_set1.tflite");
    Interpreter tflite = new Interpreter(tfliteModel);
    DataType myImageDataType = tflite.getInputTensor( inputIndex: 0).dataType();
    TensorImage tImage = new TensorImage(myImageDataType);
    tImage.load(bitmap);
    tImage = imageProcessor.process(tImage);
    float[][][][] out = new float[1][13][13][5];
    long startTimeForLoadImage = SystemClock.uptimeMillis();
    tflite.run(tImage.getBuffer(), out);
    long endTimeForLoadImage = SystemClock.uptimeMillis();
```

Deployment in Android







Conversion of Tflite Model to C Header file used in MicroProcessors

```
# This function converts a TFLite Model to C Header File used for Some MicroProcessors
def hex_to_c_array(hex_data, var_name):
  c_str = ''
  # Create header guard
  c_str += '#ifndef ' + var_name.upper() + '_H\n'
  c_str += '#define ' + var_name.upper() + '_H\n\n'
  # Add array length at top of file
  c_str += '\nunsigned int ' + var_name + '_len = ' + str(len(hex_data)) + ';\n'
  # Declare C variable
  c_str += 'unsigned char ' + var_name + '[] = {'
  hex_array = []
  for i, val in enumerate(hex_data) :
    # Construct string from hex
    hex_str = format(val, '#04x')
    # Add formatting so each line stays within 80 characters
    if (i + 1) < len(hex_data):
      hex_str += ','
    if (i + 1) % 12 == 0:
      hex_str += '\n '
    hex_array.append(hex_str)
  # Add closing brace
  c_str += '\n ' + format(' '.join(hex_array)) + '\n};\n\n'
  # Close out header guard
  c_str += '#endif //' + var_name.upper() + '_H'
  return c_str
```

```
# Write TFLite model to a C source (or header) file
c_model_name = 'model'
with open(c_model_name + '.h', 'w') as file:
  file.write(hex_to_c_array(tflite_model, c_model_name))
```

Questions?