# Model Development Phase Template

| | |
|---|---|
| Date | 12 July 2024 |
| Team ID | SWTID1720109344 |
| Project Title | Rice Type Classification using CNN |
| Maximum Marks | 10 Marks |

**Initial Model Training Code, Model Validation and Evaluation Report**:

**Initial Model Training Code (5 marks):**

Libraries importing:

```
6]: import numpy as np
    np.random.seed(42)

    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg
    import cv2
    from PIL import Image
    from skimage import io
    import scipy
    import os
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import precision_score, recall_score, f1_score, roc_curve
    import seaborn as sns
    from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
    from sklearn.model_selection import GridSearchCV
    from imblearn.over_sampling import SMOTE
    from sklearn.model_selection import StratifiedKFold
    from sklearn.model_selection import KFold

    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import load_img, img_to_array
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    import keras
    from keras import Model
    from keras.models import load_model
    from keras.utils import normalize
    from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D
    from keras.layers import Activation, Dropout, Flatten, Dense, BatchNormalization, GlobalAveragePooling2D
    from keras import layers
    from keras.utils import to_categorical
    from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, CSVLogger, LearningRateScheduler
    from keras.applications import VGG16, VGG19
```

Data splitting:

### Split the dataset

[6]:
```
X_train,X_test, y_train, y_test=train_test_split(dataset, labels, test_size=0.2, random_state=42, stratify=labels)

# Reshape = (n, image_width, image_height, n_channel)
print(X_train.shape)
print(y_train.shape)

print(X_test.shape)
print(y_test.shape)
```

```
(2000, 120, 120, 3)
(2000,)
(500, 120, 120, 3)
(500,)
```

### Normalize X values and apply one-hot encoding on y values

[7]:
```
X_train=(X_train/255)
X_test=(X_test/255)

y_train=to_categorical(y_train , num_classes=5)
y_test=to_categorical(y_test , num_classes=5)
```

## CNN-Model 1(using resnet) :

### Model building

[8]:
```
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from keras.models import Model

def create_model(input_shape):
    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3, 3), input_shape=input_shape, activation='relu')(inputs)
    x = Conv2D(32, (3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(128, (3, 3), activation='relu')(x)
    x = Conv2D(25, (1, 1))(x)

    x = Flatten()(x)

    outputs = Dense(5, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)

    return model

input_shape = (120, 120, 3)
model = create_model(input_shape)

model.summary()
```

### Train the model and save the best model

[11]:
```
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy'])

history  = model.fit(X_train, y_train,
batch_size=128,
verbose=1, epochs=25,
validation_data=(X_test, y_test),
shuffle=False, callbacks=callbacks_list)
```

## CNN -Model2(using vgg16):

[7]:
```
img_size = 224
base_model = VGG16(include_top = False,
                   weights = 'imagenet',
                   input_shape = (img_size,img_size,3))
```

## ** Training the Model**

```
try:
    history = model.fit(train,
                        epochs=50,
                        validation_data=valdata,
                        callbacks=[checkpoint, earlystopping, reduce_lr])
except KeyboardInterrupt:
    print("\nTraining Stopped")
```

CNN-Model3(using xception):

```
CNN = tf.keras.models.Sequential()
CNN.add(tf.keras.layers.Conv2D(filters = 32,kernel_size = 3, activation="relu", input_shape=[150,150,3]))
CNN.add(tf.keras.layers.MaxPool2D(pool_size = 2, strides=2))
CNN.add(tf.keras.layers.Flatten())
CNN.add(tf.keras.layers.Dense(units=512 , activation="relu"))
CNN.add(tf.keras.layers.Dense(units=5 , activation="softmax"))
CNN.summary()
```

### Training the CNN ¶

```
[14]: CNN.compile(optimizer="adam" , loss="categorical_crossentropy", metrics=['accuracy'])
      history = CNN.fit(x=train_generator, validation_data=validation_generator, epochs=10)
```

## Model Validation and Evaluation Report (5 marks):

| Model | Summary | Training and Validation Performance Metrics |
|---|---|---|
| Model 1(resnet) | Model: "functional_1"<br>Layer (type) / Output Shape / Param #<br>input_layer (InputLayer) (None, 120, 120, 3) 0<br>conv2d (Conv2D) (None, 118, 118, 32) 896<br>conv2d_1 (Conv2D) (None, 116, 116, 32) 9,248<br>max_pooling2d (MaxPooling2D) (None, 58, 58, 32) 0<br>conv2d_2 (Conv2D) (None, 56, 56, 64) 18,496<br>conv2d_3 (Conv2D) (None, 54, 54, 64) 36,928<br>conv2d_4 (Conv2D) (None, 52, 52, 64) 36,928<br>max_pooling2d_1 (MaxPooling2D) (None, 26, 26, 64) 0<br>conv2d_5 (Conv2D) (None, 24, 24, 128) 73,856<br>conv2d_6 (Conv2D) (None, 24, 24, 25) 3,225<br>flatten (Flatten) (None, 14400) 0<br>dense (Dense) (None, 5) 72,005<br>Total params: 251,582 (982.74 KB)<br>Trainable params: 251,582 (982.74 KB)<br>Non-trainable params: 0 (0.00 B) | Epoch 1/25<br>1/16 ━━━━━━━━━━━━━━━━━━━━ 6:43 27s/step - accuracy: 0.2969 - loss: 1.6098<br>WARNING: All log messages before absl::InitializeLog() is called are written to STDERR<br>I0000 00:00:1715707308.504039 70 device_compiler.h:186] Compiled cluster using XLA! This line is logged at most once for the lifetime of the process.<br>W0000 00:00:1715707308.523777 70 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.4701 - loss: 1.2660<br>W0000 00:00:1715707326.364697 70 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update<br>W0000 00:00:1715707327.457559 70 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 56s 1s/step - accuracy: 0.4865 - loss: 1.2466 - val_accuracy: 0.8140 - val_loss: 0.5017<br>Epoch 2/25<br>W0000 00:00:1715707337.199709 70 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.8469 - loss: 0.4325 - val_accuracy: 0.8940 - val_loss: 0.2885<br>Epoch 3/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.8994 - loss: 0.3003 - val_accuracy: 0.9660 - val_loss: 0.1500<br>Epoch 4/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 135ms/step - accuracy: 0.9345 - loss: 0.2223 - val_accuracy: 0.9660 - val_loss: 0.1304<br>Epoch 5/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 135ms/step - accuracy: 0.9506 - loss: 0.1505 - val_accuracy: 0.9640 - val_loss: 0.1468<br>Epoch 6/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.9604 - loss: 0.1418 - val_accuracy: 0.9520 - val_loss: 0.1647<br>Epoch 7/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.9617 - loss: 0.1172 - val_accuracy: 0.9520 - val_loss: 0.1350<br>Epoch 8/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.9718 - loss: 0.0948 - val_accuracy: 0.9520 - val_loss: 0.1640<br>Epoch 9/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.9642 - loss: 0.0954 - val_accuracy: 0.9460 - val_loss: 0.1809<br>Epoch 10/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 135ms/step - accuracy: 0.9608 - loss: 0.1057 - val_accuracy: 0.9520 - val_loss: 0.1549<br>Epoch 11/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 136ms/step - accuracy: 0.9680 - loss: 0.1163 - val_accuracy: 0.9520 - val_loss: 0.1576<br>Epoch 12/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 136ms/step - accuracy: 0.9657 - loss: 0.0754 - val_accuracy: 0.9460 - val_loss: 0.2350<br>Epoch 13/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 138ms/step - accuracy: 0.9803 - loss: 0.0756 - val_accuracy: 0.9260 - val_loss: 0.2195<br>Epoch 14/25<br>16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 140ms/step - accuracy: 0.9724 - loss: 0.0887 - val_accuracy: 0.9480 - val_loss: 0.1668 |

<table>
<tr>
<td>Model 2<br><br>(using vgg)</td>
<td>
Model: "sequential"<br>
_____<br>
Layer (type) Output Shape Param #<br>
==========<br>
vgg16 (Functional) (None, 7, 7, 512) 14714688<br>
global_average_pooling2d ( (None, 512) 0<br>
GlobalAveragePooling2D)<br>
dense (Dense) (None, 512) 262656<br>
dropout (Dropout) (None, 512) 0<br>
dense_1 (Dense) (None, 5) 2565<br>
==========<br>
Total params: 14979909 (57.14 MB)<br>
Trainable params: 7344645 (28.02 MB)<br>
Non-trainable params: 7635264 (29.13 MB)<br>
_____
</td>
<td>
Epoch 1/50<br>
938/938 [==========] - ETA: 0s - loss: 0.1043 - accuracy: 0.9623<br>
Epoch 1: val_loss improved from inf to 0.02687, saving model to model.h5<br>
938/938 [==========] - 1265s 1s/step - loss: 0.1043 - accuracy: 0.9623 - val_loss: 0.0269 - val_accuracy: 0.9911 - lr: 1.0000e-04<br>
Epoch 2/50<br>
938/938 [==========] - ETA: 0s - loss: 0.0348 - accuracy: 0.9881<br>
Epoch 2: val_loss improved from 0.02687 to 0.02138, saving model to model.h5<br>
938/938 [==========] - 990s 1s/step - loss: 0.0348 - accuracy: 0.9881 - val_loss: 0.0214 - val_accuracy: 0.9923 - lr: 1.0000e-04<br>
Epoch 3/50<br>
938/938 [==========] - ETA: 0s - loss: 0.0291 - accuracy: 0.9904<br>
Epoch 3: val_loss improved from 0.02138 to 0.01031, saving model to model.h5<br>
938/938 [==========] - 991s 1s/step - loss: 0.0291 - accuracy: 0.9904 - val_loss: 0.0103 - val_accuracy: 0.9966 - lr: 1.0000e-04<br>
Epoch 4/50<br>
938/938 [==========] - ETA: 0s - loss: 0.0250 - accuracy: 0.9924<br>
Epoch 4: val_loss did not improve from 0.01031<br>
938/938 [==========] - 992s 1s/step - loss: 0.0250 - accuracy: 0.9924 - val_loss: 0.0146 - val_accuracy: 0.9955 - lr: 1.0000e-04<br>
Epoch 5/50
</td>
</tr>
<tr>
<td>Model 3<br><br>(xception)</td>
<td>
Model: "sequential"<br>
<table>
<tr><th>Layer (type)</th><th>Output Shape</th><th>Param #</th></tr>
<tr><td>conv2d (Conv2D)</td><td>(None, 148, 148, 32)</td><td>896</td></tr>
<tr><td>max_pooling2d (MaxPooling2D)</td><td>(None, 74, 74, 32)</td><td>0</td></tr>
<tr><td>flatten (Flatten)</td><td>(None, 175232)</td><td>0</td></tr>
<tr><td>dense (Dense)</td><td>(None, 512)</td><td>89,719,296</td></tr>
<tr><td>dense_1 (Dense)</td><td>(None, 5)</td><td>2,565</td></tr>
</table>
Total params: 89,722,757 (342.27 MB)<br>
Trainable params: 89,722,757 (342.27 MB)<br>
Non-trainable params: 0 (0.00 B)
</td>
<td>
Epoch 1/10<br>
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR<br>
I0000 00:00:1716249113.986925 119 service.cc:145] XLA service 0x7928ec00c040 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:<br>
I0000 00:00:1716249113.986977 119 service.cc:153] StreamExecutor device (0): Tesla T4, Compute Capability 7.5<br>
I0000 00:00:1716249113.986983 119 service.cc:153] StreamExecutor device (1): Tesla T4, Compute Capability 7.5<br>
7/3282 ———— 1:16 23ms/step - accuracy: 0.2554 - loss: 6.1027<br>
I0000 00:00:1716249118.539742 119 device_compiler.h:188] Compiled cluster using XLA! This line is logged at most once for the lifetime of the process.<br>
3282/3282 ———— 91s 26ms/step - accuracy: 0.9291 - loss: 0.2791 - val_accuracy: 0.9669 - val_loss: 0.0982<br>
Epoch 2/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9777 - loss: 0.0657 - val_accuracy: 0.9724 - val_loss: 0.0936<br>
Epoch 3/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9875 - loss: 0.0357 - val_accuracy: 0.9756 - val_loss: 0.0888<br>
Epoch 4/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9943 - loss: 0.0159 - val_accuracy: 0.9677 - val_loss: 0.1541<br>
Epoch 5/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9975 - loss: 0.0083 - val_accuracy: 0.9711 - val_loss: 0.1500<br>
Epoch 6/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9983 - loss: 0.0045 - val_accuracy: 0.9691 - val_loss: 0.1904<br>
Epoch 7/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9988 - loss: 0.0036 - val_accuracy: 0.9740 - val_loss: 0.2000<br>
Epoch 8/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9993 - loss: 0.0024 - val_accuracy: 0.9705 - val_loss: 0.2450<br>
Epoch 9/10<br>
3282/3282 ———— 142s 25ms/step - accuracy: 0.9991 - loss: 0.0027 - val_accuracy: 0.9738 - val_loss: 0.2122<br>
Epoch 10/10<br>
3282/3282 ———— 83s 25ms/step - accuracy: 0.9996 - loss: 0.0011 - val_accuracy: 0.9735 - val_loss: 0.2350
</td>
</tr>
</table>