# Model Development Phase Template

| | |
|---|---|
| Date | 29 October 2024 |
| Team ID | SWTID1727274979 |
| Project Title | Deep learning techniques for breast cancer risk prediction |
| Maximum Marks | 10 Marks |

## Initial Model Training Code, Model Validation and Evaluation Report

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

**Step 1 :** Importing necessary libraries

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
```

**Step 2 :** Downloading and preparing the dataset

```
!pip install -q kaggle
!mkdir ~/.kaggle
!cp /content/kaggle.json ~/.kaggle/
!kaggle datasets download -d paultimothymooney/breast-histopathology-images
```

**Step 3 :** Unzipping the dataset

```
import zipfile
!unzip /content/breast-histopathology-images.zip
```

**Step 4 :** Importing the ImageDataGenerator to load and augment images

"12932" folder contains balanced data(images) to some extent for training part .

```
data_dir = '/content/12932'
```

Importing ImageDataGenerator to load image :

```
# Image data generator for loading and augmenting images
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

val_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

**Step 5 :** Building the CNN model by adding necessary layers

```
# Building The Model by Adding Necessary Layers
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),     #Adding Convolutional layer
    MaxPooling2D(pool_size=(2, 2)),    # Adding MaxPooling Layer
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),                # Adding Flatten Layer
    Dense(128, activation='relu'),     # Adding Dense Layer
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

**Step 6 :** Compiling the model

```
#Compilimg The Model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

**Step 7 :** Training and fitting the model

```
# Training and fitting the model , using Earlystopping to avoid model Overfitting

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)]
)
```

**Step 8 :** Evaluating the model

```
# Printing Accuracy
val_loss, val_accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

**Step 9 :** Plotting training and validation accuracy and loss

```
# Plotting Accuracy and Validation Loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

**Step 10 :** Saving the model

```
model.save("breastcancer.h5")
```

- **Initial Model Testing Code :**

**Step 1 :** Importing the necessary libraries

```
[ ]  # Importing necessary packages
     import numpy as np
     import matplotlib.pyplot as plt
     from keras.models import load_model
     from keras.preprocessing import image
     from keras.layers import Flatten, Reshape
```

**Step 2 :** Loading The Model

```
# Loading the saved model as breastcancer.h5
model = load_model('breastcancer.h5')
```

Specifying the path to dataset :-

```
[ ]  # Specifying the path to dataset
     data_dir = '12932'
     batch_size = 32
     img_size = (128, 128)
```

**Step 3 :** Printing Sample of Images Belonging to Benign and Malignant classes

```
import os
print("Benign samples:", len(os.listdir('/content/12932/0')))
print("Malignant samples:", len(os.listdir('/content/12932/1')))
```

```
Benign samples: 433
Malignant samples: 304
```

**Step 4 :** Preprocessing The Image

```
# Preprocessing the Images
from tensorflow.keras.preprocessing import image

def predict_image(img_path):
    img = image.load_img(img_path, target_size=(128, 128))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    return "Malignant" if prediction > 0.5 else "Benign"
```

**Step 5 :** Making Predictions for the test Image Belonging to class 0

```
[ ] # Making prediction for the test image for the class 0

    # Path of image  is mentioned for prediction
    print(predict_image('/content/12242/0/12242_idx5_x1001_y251_class0.png'))

    1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
    Benign
```

Step 6 : Making Predictions for the test Image Belonging to class 1

```
[ ] # Making prediction for the test image for the class 1

    # Path of image  is mentioned for prediction
    print(predict_image('/content/12932/1/12932_idx5_x1001_y1251_class1.png'))

    1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
    Malignant
```

Step 7 : Taking Any random Sample image from the Dataset To see whether the Model is predicting right or not

```
[ ] # Setting the path and predicting the image class

    print(predict_image('/content/12876/0/12876_idx5_x1201_y51_class0.png'))

    1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
    Benign
```

**Conclusion :**

THE MODEL IS TRAINED AND TESTED CORRECTLY AND PREDICTING RIGHT CLASSES OF IMAGES .

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

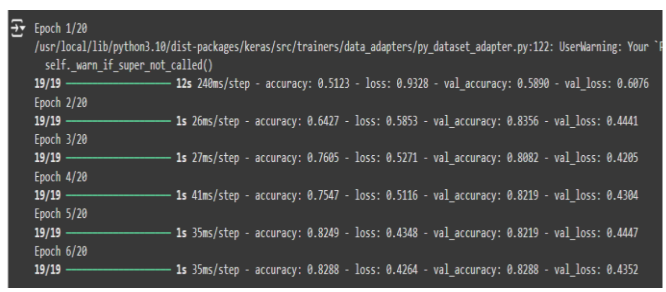
A Brief Summary for the model Validation and Evaluation is presented in the following table which includes the following :

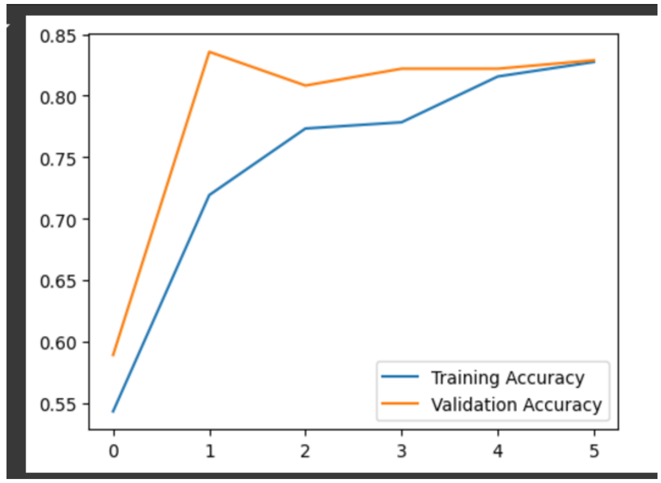
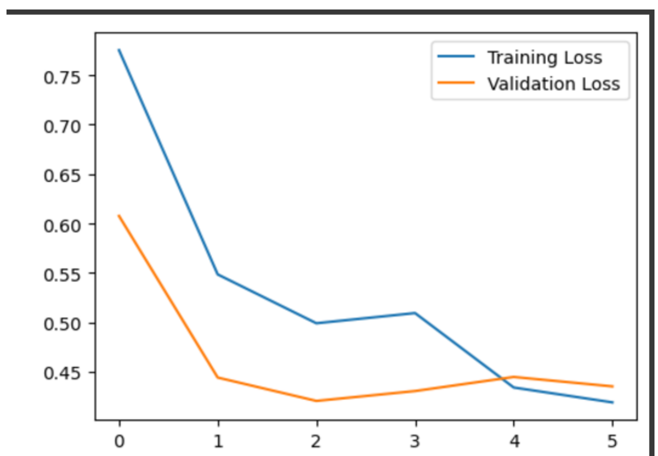
1) Training Epochs Summary .

2) Visualization plot of Training and Validation Accuracy .

3) Visualization plot of Training and Validation Loss .

4) Accuracy Obtained .

| **Model** | **Summary** | **Training and Validation Performance Metrics** |
|---|---|---|
| Model 1<br>( CNN ) | **1) Epochs Summary ( EarlyStopping is used to avoid overfitting ) –**<br>• The training accuracy steadily increases from **0.5123** to **0.8288** over the **6 epochs,** while the training loss consistently decreases. This suggests that the model is learning effectively from the training data .<br>• The validation accuracy initially increases to **0.8882** in **epoch 3** but then fluctuates slightly. The validation loss also decreases initially but then stabilizes. This indicates that the model is generalizing well to unseen data and is not overfitting significantly.<br>**EarlyStopping :**<br>Early stopping is a technique that helps prevent overfitting by automatically stopping the training process when the model's performance on a validation set starts to deteriorate.<br>**Key Benefit of earlystopping :** | <br>Epoch 1/20<br>/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `P<br>self._warn_if_super_not_called()<br>19/19 ━━━━━━━━━━ 12s 240ms/step - accuracy: 0.5123 - loss: 0.9328 - val_accuracy: 0.5890 - val_loss: 0.6076<br>Epoch 2/20<br>19/19 ━━━━━━━━━━ 1s 26ms/step - accuracy: 0.6427 - loss: 0.5853 - val_accuracy: 0.8356 - val_loss: 0.4441<br>Epoch 3/20<br>19/19 ━━━━━━━━━━ 1s 27ms/step - accuracy: 0.7605 - loss: 0.5271 - val_accuracy: 0.8082 - val_loss: 0.4205<br>Epoch 4/20<br>19/19 ━━━━━━━━━━ 1s 41ms/step - accuracy: 0.7547 - loss: 0.5116 - val_accuracy: 0.8219 - val_loss: 0.4304<br>Epoch 5/20<br>19/19 ━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8249 - loss: 0.4348 - val_accuracy: 0.8219 - val_loss: 0.4447<br>Epoch 6/20<br>19/19 ━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8288 - loss: 0.4264 - val_accuracy: 0.8288 - val_loss: 0.4352 |

| | | |
|---|---|---|
| | **a) Prevents Overfitting :**<br><br>By stopping training early, we can avoid the model memorizing the training data and improve its generalization ability . | |
| | **2) Training and Validation Accuracy Plot** – A training and validation accuracy plot is a visual representation of how well a model performs . This plot is a crucial tool for understanding and optimizing model performance.<br><br>**• Training Accuracy :**<br><br>Measures the model's accuracy on the training dataset .<br><br>**• Validation Accuracy :**<br><br>Measures the model's accuracy on a separate validation dataset , which the model hasn't seen during training . |  |
| | **3) Training and Validation Loss Plot –**<br><br>A training and validation loss plot is a visual representation of how well a model is learning and generalizing during the training process.<br><br>**• Training Loss :**<br><br>Measures the error the model makes on the training data .<br><br>**• Validation Loss :**<br><br>Measures the error the model makes on a separate validation dataset, which the model hasn't seen during training .<br><br>This is a crucial metric to assess the model's ability to generalize to new, unseen data . |  |

| | | |
|---|---|---|
| | **4) Obtained accuracy after training and testing the CNN Model – 82.88 %** | `# Printing Accuracy`<br>`val_loss, val_accuracy = model.evaluate(val_generator)`<br>`print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")`<br><br>`5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8318 - loss: 0.4277`<br>`Validation Accuracy: 82.88%` |