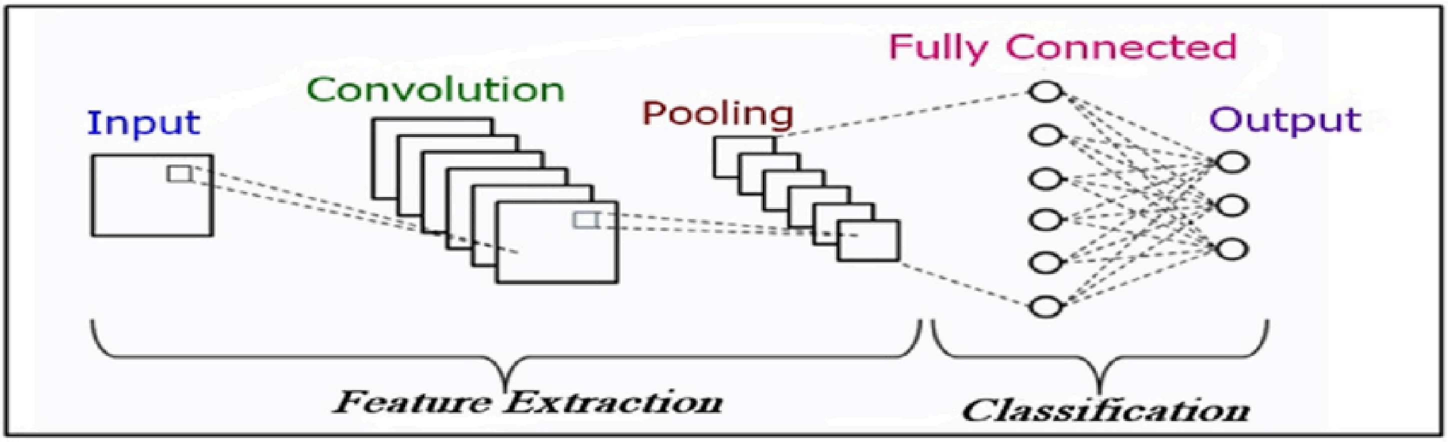
Convolutional Neural Networks



Hemorrhage Detection

Introduction

In this project, I have used a convolutional neural network architecture for classification of patients suffering from hemorrhage by using head CT images. Talking about the dataset that I have used for this project consists of healthy patients which are classified as "Normal" and un healthy patients are classified as "Hemorrhage".

Libraries Required

- Python 3.6+
- glob (pip install glob)
- Keras (pip install keras)
- NumPy (pip install numpy)
- Pandas (pip install pandas)
- Seaborn (pip install seaborn)
- MatplotLib (pip install matplotlib)
- Tensorflow (pip install tensorflow)
- Scikit-learn (pip install scikit-learn)
- Operating System (built-in library, use "import os")

Data Processing

First step here is to generate data, I have jpeg images of the head CT. To make sure all images which are going to fed to the network are pre-processed for it to learn and analyze those images accurately.

We need to import *ImageDataGenerator* from Keras library.

```
#Import necessary libraries
from keras.preprocessing.image import ImageDataGenerator
```

First step here is to do some data augmentation in which we can rescale, apply shear transformation, apply rotation to the image.

```
# Define ImageDataGenerator for training data with augmentation
Generator = ImageDataGenerator(
    rescale=1./255,          # Rescale pixel values to the range [0,1]
    zoom_range=0.25,         # Randomly zoom images by 25%
    shear_range=0.25,        # Apply shear transformation with max intensity of 25%
    rotation_range=25,       # Randomly rotate images up to 25 degrees
    horizontal_flip=True,    # Randomly flip images horizontally
    fill_mode="nearest",     # Fill in newly created pixels after rotation or shifting with nea
    validation_split=0.15    # Split the data into training and validation sets, with 15% of th
)

# Define ImageDataGenerator for testing data without augmentation
Test_Generator = ImageDataGenerator(
    rescale=1./255          # Only rescale pixel values for testing data
)
```

Now, we need to generate training, testing and validating dataset for convolutional network.

```
# Generate flow of training images from dataframe with specified parameters
Train_IMG_Set = Generator.flow_from_dataframe(
    dataframe=Train_Data,       # DataFrame containing training data file paths and correspondi
    x_col="JPG",                # Column in DataFrame containing file paths to images
    y_col="CATEGORY",           # Column in DataFrame containing labels for images
    color_mode="grayscale",     # Convert images to grayscale
    class_mode="categorical",   # Type of labels; in this case, categorical labels (one-hot enc
    subset="training"           # Subset of data to use; in this case, the training subset
)

# Generate flow of validation images from dataframe with specified parameters
Validation_IMG_Set = Generator.flow_from_dataframe(
    dataframe=Train_Data,       # DataFrame containing training data file paths and correspondi
    x_col="JPG",                # Column in DataFrame containing file paths to images
    y_col="CATEGORY",           # Column in DataFrame containing labels for images
    color_mode="grayscale",     # Convert images to grayscale
    class_mode="categorical",   # Type of labels; in this case, categorical labels (one-hot enc
    subset="validation"         # Subset of data to use; in this case, the validation subset
)

# Generate flow of test images from dataframe with specified parameters
Test_IMG_Set = Generator.flow_from_dataframe(
    dataframe=Test_Data,        # DataFrame containing test data file paths and corresponding la
    x_col="JPG",                # Column in DataFrame containing file paths to images
    y_col="CATEGORY",           # Column in DataFrame containing labels for images
    color_mode="grayscale",     # Convert images to grayscale
    class_mode="categorical"    # Type of labels; in this case, categorical labels (one-hot enc
)
```

Here, we are using pre defined *'Generator'* to generated augmented images for *'Training'* and *'Validation'*. For *'Testing'* we are using the same generator but without any image augmentation.

CNN Model Architecture

We first need to initialize a sequential model to create an *CNN architecture*.

```
Model = Sequential()
```

We will now build model architecture using convolutional layers, dropout, batchnormalization, and max-pooling.

```
Model.add(Conv2D(12,(3,3),activation="relu",input_shape=(256,256,1)))   # adds 2D conv layer wi
Model.add(BatchNormalization())                                        # adds batchnormalizatio
Model.add(MaxPooling2D((2,2)))                                         # adds max-pooling layer
Model.add(Conv2D(24,(3,3),activation="relu",padding="same"))           # adds another 2D conv l
Model.add(Dropout(0.2))                                                # adds dropout rate of (
Model.add(MaxPooling2D((2,2)))                                         # adds another max-pooli
Model.add(Conv2D(64,(3,3),activation="relu",padding="same"))           # adds another 2D conv l
Model.add(Dropout(0.5))                                                # adds dropout rate of (
Model.add(MaxPooling2D((2,2)))                                         # adds another max-pooli
Model.add(TimeDistributed(Flatten()))                                  # adds flatten layer to
Model.add(Flatten())                                                   # this will convert 3D t
Model.add(Dense(256, activation="relu"))                               # adds fully connected D
Model.add(Dropout(0.5))                                                # adds another dropout r
Model.add(Dense(2, activation="softmax"))                              # final dense (output) l
```

Next step here is to train this CNN architecture using the training, testing and validation datset which we generated previously.

We need to employ a callback which can keep monitoring the training loss throughout and stops training if the loss value does not improve after a certain number of epochs.

```
Call_Back = tf.keras.callbacks.EarlyStopping(monitor="loss", patience=5, mode="min")
```

We need to compile this by selecting the appropriate optimizer, loss function to calculate loss, and evalution metrics for the CNN model. Then, using *model.fit* we ae going to train the CNN model on the training dataset.

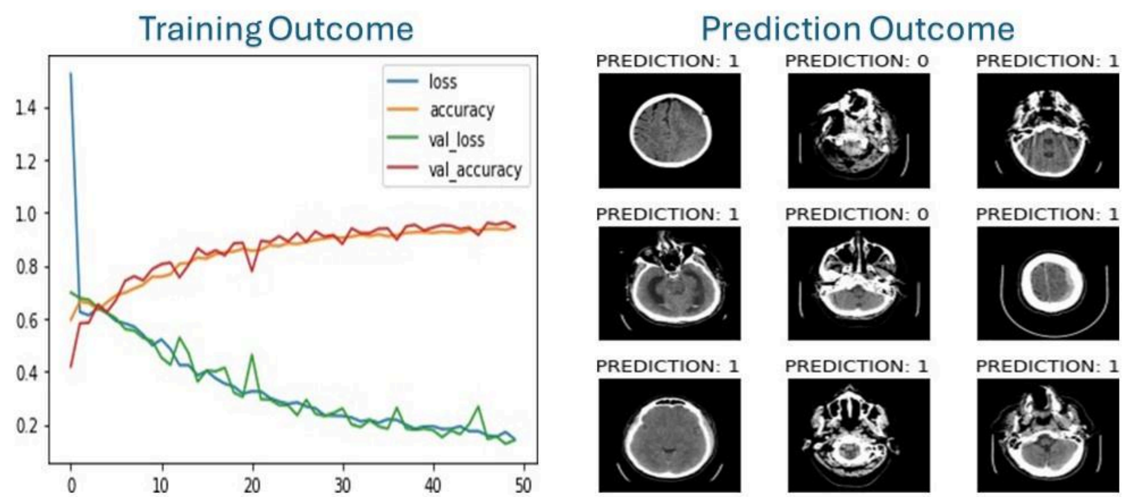
```
# Compiling the CNN model
Model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
# Training the CNN model
CNN_Model = Model.fit(Train_IMG_Set,
                    validation_data=Validation_IMG_Set, callbacks=Call_Back,
                    batch_size=32,
                    epochs=50)
```

To evaluate the performance of CNN model we going to use *model.evaluate* on the testing dataset that we generated previously. Lastly, I have used *mdoel.predict* to test the efficiency of the trained CNN model on *Testing Dataset*

```
# Evaluate the performance
Model_Results = Model.evaluate(Test_IMG_Set, verbose=False)
# Predicting using the trained model
Prediction = Model.predict(Test_IMG_Set)
Prediction = Prediction.argmax(axis=-1)
```

Result

After training and evaluating the CNN model over the differnet training parameters, model is 96% accurate in predicting/classifing the hemorrhage using head CT images with the training loss of *'0.1184'*.



Discusion

Overall, this CNN model is able to classify/detect hemorrahge by using *Head CT* images with an accuracy of around 96%.