Another trial using Keras

We also decided to experiment with functionalities of Keras.

We inputted the pre-processed images using the Keras library in tensorflow, and use its modules like Imagedatagenerator and ".flow_from_directory" function.

We normalize the pixel values using 'rescale' in the 'Imagedatagenerator' module, using the '.flow_from_directory', we classified it as categorical,converted the images into grayscal, shuffled the batches for the training data set and did data augmentation for the training dataset, i.e, randomly applied shear, zoomed in, flipped, shifted and rotated the images to have more images for training our model.

Since the training, testing and validation set were separated earlier we had to do it to each of them by calling their respective directories.

We experimented with the batch sizes over here, which did show an effect and a size of 64 was found to be optimal

```
train_dir='dataset_final/train_Prepro'
val_dir='dataset_final/val_Prepro'
test_dir='dataset_final/test_Prepro'
```

```
batch_size=64
from keras.preprocessing.image import ImageDataGenerator

# Define the data augmentation and preprocessing pipeline
train_datagen = ImageDataGenerator(
    rescale=1./255,
rotation_range=20,  # randomly rotate the images by up to 20 degrees
    width_shift_range=0.2,  # randomly shift the images horizontally by up to 20% of the width
    height_shift_range=0.2,  # randomly shift the images vertically by up to 20% of the height
    shear_range=0.2,  # randomly apply shearing transformations
    zoom_range=0.2,  # randomly zoom in or out on the images
    horizontal_flip=True,  # randomly flip the images horizontally
    fill_mode='nearest'  # fill in any missing pixels with the nearest available pixel
)

# Create the data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode="grayscale",
    shuffle=True  # shuffle the data at the beginning of each epoch
)

val_datagen = ImageDataGenerator(rescale=1./255)
val_generator = val_datagen.flow_from_directory(
    val_dir,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode="grayscale",
    shuffle=False  # don't shuffle the validation data
)

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode="grayscale",
    shuffle=False  # don't shuffle the test data
)
```

```
Found 7461 images belonging to 6 classes.
Found 1595 images belonging to 6 classes.
Found 1605 images belonging to 6 classes.
```

Creating a model in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
```

Sequential is a class that allows us to create a neural network model where layers are added one after another.

Conv2D and MaxPooling2D are classes that represent 2D convolutional and max pooling layers respectively. Convolutional layers are often used in computer vision tasks to extract features

from images, and max pooling layers are used to downsample the feature maps produced by the convolutional layers.

Dense is a class that represents a fully connected layer. A fully connected layer connects all the neurons in the previous layer to all the neurons in the current layer.

Flatten is a class that flattens the output of the previous layer into a 1D array, which can be passed to a fully connected layer.

Dropout is a class that represents a regularization technique used in neural networks to prevent overfitting. It randomly drops out a fraction of the neurons in the previous layer during training, forcing the network to learn more robust features, which was proved to be very useful.

MODEL

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 16)      160

max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
)

dropout (Dropout)            (None, 127, 127, 16)      0

conv2d_1 (Conv2D)            (None, 125, 125, 64)      9280

max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
2D)

conv2d_2 (Conv2D)            (None, 60, 60, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0
2D)

dropout_1 (Dropout)          (None, 30, 30, 128)       0

flatten (Flatten)            (None, 115200)            0

dense (Dense)                (None, 512)               58982912

dropout_2 (Dropout)          (None, 512)               0

dense_1 (Dense)              (None, 6)                 3078

=================================================================
Total params: 59,069,286
Trainable params: 59,069,286
Non-trainable params: 0
_________________________________________________________________
```

```
model = Sequential()
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,1)))
model.add(MaxPooling2D())
model.add(Dropout(0.15)
model.add(Conv2D(64, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(128, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.15))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(6, activation='softmax'))
```

We decided to use this model, with so many layers, since less complex models couldn't fit the training data.
Without dropout the model was overfitting by a lot.

All layers have the activation function ‘relu’ but the dense layer has ‘softmax’ since we need a multi-classification and softmax fits the use case.

```
model.compile('adam' , loss=tf.losses.CategoricalCrossentropy(), metrics=['accuracy'])
```

We use the ‘adam’ optimizer which is quite popular for classification problems with the ‘categoricalcrossentropy’ loss function.

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

TRAINING THE MODEL

```
logdir='logs'
```

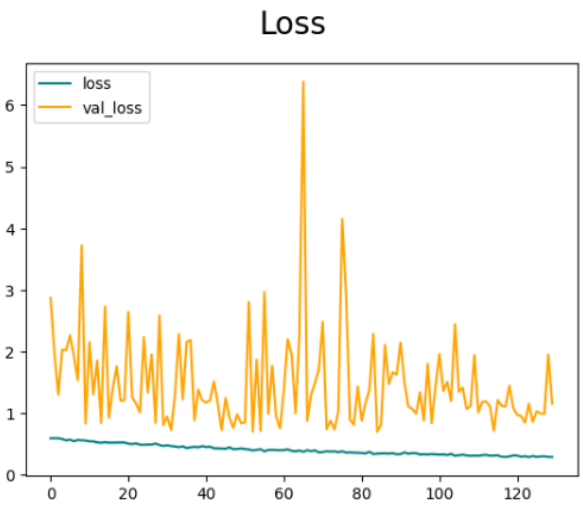
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

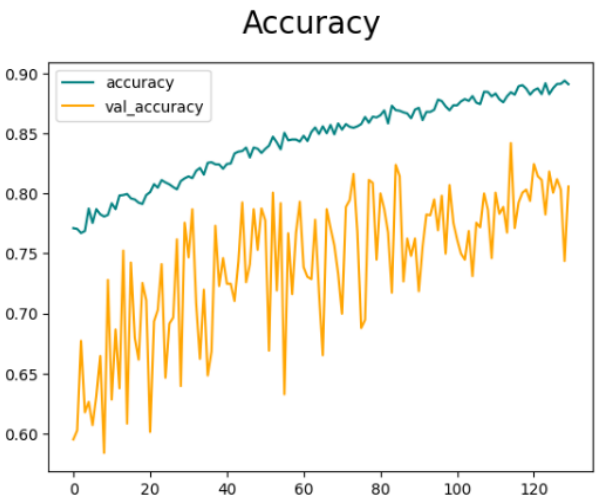
This utilises TensorBoard callback in Keras, which will write summary logs to the 'logs' directory during training. TensorBoard is a visualization tool that can be used to monitor the training and performance of deep learning models, and the logs can be used to visualize the training progress in TensorBoard.

```
hist = model.fit(train_generator, epochs=130, validation_data=val_generator, callbacks=[tensorboard_callback])
```

Using the fit() function in Keras, which takes in the training data (train_generator), validation data (val_generator), and the number of epochs to train for (130). During training, the tensorboard_callback function is called, which writes summary logs to the 'logs' directory for visualization in TensorBoard. The fit() function returns a history object, hist, which contains information about the training process. This object can be used to analyze the performance of the model and make decisions about how to adjust the training parameters.

The training visualised:





Loss vs epochs

Accuracy vs epochs

SAVING THE MODEL

```
from keras.models import load_model

# Save the trained model with custom name "Keras"
model.save("Keras.h5")

# Load the saved model
loaded_model = load_model("Keras.h5")
```

We save the model in h5 format with the name “keras”. Saving a model in H5 format allows you to save the entire model architecture, including the weights and optimizer state, in a single binary file.

TESTING THE MODEL

```
# Test the model on the testing data
test_loss, test_accuracy = model.evaluate(test_generator)

print("Test loss:", test_loss)
print("Test accuracy:", test_accuracy)
```

```
26/26 [==============================] - 2s 83ms/step - loss: 1.1991 - accuracy: 0.7938
Test loss: 1.1990758180618286
Test accuracy: 0.7937694787979126
```

We test the model on the normalized testing data from earlier.