Task 1: Regression

Data loading

Data loading

```
In [21]: # the target is  to predict the levelCancerAntigen
target = 'levelCancerAntigen'
features = ['logCancerVol', 'logCancerWeight', 'age', 'logBenighHP',
            'svi', 'logCP', 'gleasonScore', 'gleasonS45']
# Loading data
df = pd.read_csv('Task1_RegressionTask_CancerData.csv')
df.head()
```

Out[21]:

	index	logCancerVol	logCancerWeight	age	logBenighHP	svi	logCP	gleasonScore	gleason
0	1	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	
1	2	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	
2	3	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	
3	4	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	
4	5	0.751416	3.432373	62	-1.386294	0	-1.386294	6	

Figure [1] : Data loading

From figure [1] ,The dataset used for the regression task is loaded from the 'Task1_RegressionTask_CancerData.csv' filein which dataset contains information related to cancer cases and aiming to predict the 'levelCancerAntigen' based on a set of features , hence the first step is the target Variable and features Selection where the target variable aiming to predict is 'levelCancerAntigen.' And the selected set of features to use for the regression task. These features include:

- 'logCancerVol'
- 'logCancerWeight'
- 'age'
- 'logBenighHP'
- 'svi'
- 'logCP'
- 'gleasonScore'
- 'gleasonS45'

then the data is loaded from the file into a Pandas DataFrame using the **pd.read_csv** function. This function reads the data from the CSV file and stores it in the DataFrame named 'df.' Moreover , to get an initial overview of the dataset, the first few rows using the **df.head()** method are desplayed. This provides an overview of the data structure and the values of the selected features, including the target variable.

Data preprocessing

```
In [77]: from sklearn.model_selection import train_test_split
         # Split the data into trainSing and testing sets
         X, y = df[features].values, df[target].values
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [78]: # showing the The correlation matrix
         df[is_train][features].corr()
```

Figure [2] : Implementing train test split and showing the correlation matrix.

As it shown from Figure [2] , The dataset is divided into training and test sets based on the 'train' flag. For the training set, 'X_train' represents the feature values, including 'logCancerVol,' 'logCancerWeight,' 'age,' 'logBenighHP,' 'svi,' 'logCP,' 'gleasonScore,' and 'gleasonS45.' The corresponding 'levelCancerAntigen' values are stored in 'y_train.' The test set consists of 'X_test' and 'y_test,' containing the same features and 'levelCancerAntigen' values, respectively.

Moreover, An analysis of the correlation matrix of the selected predictors is performed to understand the relationships between the variables. The correlation matrix provides insights into the strength and direction of the relationships between each pair of variables. Strong correlations can cause issues, which can affect the performance of the regression model.

```
In [80]: from sklearn.preprocessing import StandardScaler
         import statsmodels.api as sm

In [81]: # standardize each numerical variable in the dataset to ensure that each variable has a similar scale
         scaler = StandardScaler().fit(X)
         X_train = scaler.transform(X_train)
         X_test = scaler.transform(X_test)
```

Figure [4] : Standardize data.

An important aspect to fitting the regression model, the selected features are standardized to have unit variance as shown in figure [4]. Standardization is a common preprocessing step that ensures that all predictors have the same scale. This step is essential when working with models sensitive to the scale of features, such as least square regression. The 'StandardScaler' from scikit-learn is used to standardize the features hence with the data split into training and test sets and the features standardized, the dataset is now prepared for modeling.

Regression Models

Least Square regression

```
In [83]: y_hat = ls.predict(sm.add_constant(X_test))
         ls_error_rate = mean_squared_error(y_test, y_hat)
         ls_std_error = np.std((y_test - y_hat)**2, ddof=1)/np.sqrt(y_test.size)
         print(f'Least Squares Test Error: {ls_error_rate:.3f}')
         print(f'               Std Error: {ls_std_error:.3f}')

         Least Squares Test Error: 0.347
                        Std Error: 0.138
```

Figure [5] : Part of Least square regression

From figure[5], Least Square (LS) model is applied to the standardized features, and its performance is assessed on the test dataset the LS model's predictions on the test data are denoted as 'y_hat.' The evaluation metrics for the LS model are as follows:

Test Error Rate: The mean squared error (MSE) on the test set is a measure of the model's prediction accuracy. The MSE is a metric that quantifies the average squared difference between the model's predictions and the actual target values. In this case, the LS model achieves a test error rate of approximately 0.347.

Standard Error: The standard error, calculated as the standard deviation of the squared differences between the true and predicted values, is used to estimate the model's variability. The standard error is a measure of the uncertainty associated with the model's predictions. For the LS model, the standard error is approximately 0.138.

These metrics provide valuable insights into the performance and reliability of the LS model in predicting the 'levelCancerAntigen' based on the selected features.

K-folds cross validation

```python
def train_cv(X_sub, y_sub, n_splits=5, shuffle=True, model_type='lasso'):
    #a K-Fold cross-validator
    k_fold = KFold(n_splits=n_splits, shuffle=shuffle)

    #  the current alpha value
    curr_alpha = 0.00001  # Alpha range: 0 -> 1
    # Lists to store alpha values and Mean Squared Errors (MSE)
    alpha_list = []
    MSE = []

    for train_index, test_index in k_fold.split(X_sub):
        # Split data into training and testing subsets
        X_train_sub, X_test_sub = X_sub[train_index], X_sub[test_index]
        y_train_sub, y_test_sub = y_sub[train_index], y_sub[test_index]

        if model_type == 'lasso':
            #  a Lasso regression model
            reg_model = Lasso()
        elif model_type == 'ridge':
            # a Ridge regression model
            reg_model = Ridge()
        else:
            raise ValueError("Invalid model_type. Use 'lasso' or 'ridge'.")

        # Loop through different alpha values and train the model
        while curr_alpha <= 1:  # Alpha range: 0 -> 1
            reg_model.set_params(alpha=curr_alpha)
            alpha_list.append(curr_alpha)
            reg_model.fit(X_train_sub, y_train_sub)
            y_pred_sub = reg_model.predict(X_test_sub)
            MSE.append(mean_squared_error(y_test_sub, y_pred_sub))
            curr_alpha += 0.00001

    return MSE, alpha_list
```

Figure [6] : Implementing K-Folds

The function, `train_cv`, shown in figure [6] ,trains and cross-validates Lasso or Ridge regression models using input data `X_sub` and target values `y_sub` it also allows customization of cross-validation parameters and the choice of regression model. The function outputs a list of MSE values and the corresponding alpha values used in the cross-validation. The following steps summarize the K-folds functionality :

1. It initializes a K-Fold cross-validator with the specified number of splits and shuffle option.
2. It initializes a variable **curr_alpha** to a small alpha value (0.00001) and creates empty lists to store alpha values and MSE values.
3. It enters a loop to iterate through different alpha values for the chosen model (Lasso or Ridge). The loop continues until the alpha value reaches 1.0.
4. For each alpha value, the function:
    a. Creates a new instance of the regression model with the current alpha value.
    b. Fits the model on the training subset.
    c. Predicts the target values for the test subset.
    d. Calculates the Mean Squared Error (MSE) between the predicted and actual target values for that alpha value.

e. Appends the alpha value and corresponding MSE to the respective lists.
f. Increments the alpha value by a small step (0.00001) for the next iteration.

5. After the loop, the function returns the list of MSE values and the corresponding alpha values.

this function is used to perform cross-validation and select the optimal alpha value for Lasso or Ridge regression.

Lasso Regression Model

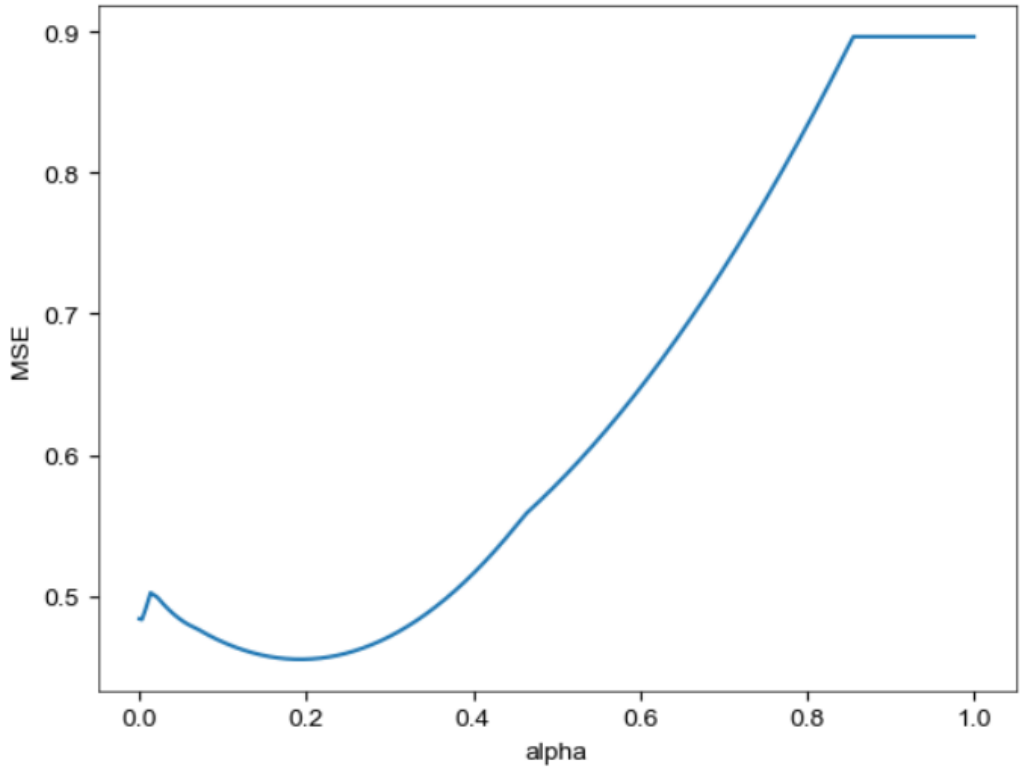


Figure [7] : Graph indicating the best alpha value for lasso regression

```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

X_train2, X_test2, y_train2, y_test2 = train_test_split(X_test, y_test, test_size=0.9, random_state=42)

reg_model = Lasso(alpha=best_alpha)
reg_model.fit(X_train, y_train)
y_pred_new = reg_model.predict(X_test2)

print(reg_model.alpha)

print("Mean squared error: %.2f" % mean_squared_error(y_test2, y_pred_new))

print('R2 score: %.2f' % r2_score(y_test2, y_pred_new))
```

```
0.19282000000005206
Mean squared error: 0.59
R2 score: 0.62
```

Figure [8] : snipt of lasso regression model

The Lasso regression model is applied as shown in figure [8] to predict 'levelCancerAntigen' on the test dataset. It's initialized with an optimal alpha value, best_alpha, shown in figure [7]. The model is effectively fitted to the training data, and its predictive performance is measured through the generated predictions (y_pred_new) on the test data (X_test2). also the optimal alpha value is reported as 0.19282000000005206, a parameter that influences regularization and feature selection in the model. The Lasso model provide strong predictive accuracy, with a Mean Squared Error (MSE) of 0.59. Additionally, it explains a significant proportion of the variance in the target variable, achieving an R-squared (R2) score of approximately 0.62.

Ridge regression

```
In [152]: # create a Ridge regression
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

X_train3, X_test3, y_train3, y_test3 = train_test_split(X_test, y_test, test_size=0.9, random_state=42)

reg_model = Ridge(alpha=best_alpha2)
reg_model.fit(X_train, y_train)
y_pred_new2 = reg_model.predict(X_test3)

print(reg_model.alpha)

# print the mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test3, y_pred_new2))

# print the R2 score
print('R2 score: %.2f' % r2_score(y_test3, y_pred_new2))
```

```
1e-05
Mean squared error: 0.38
R2 score: 0.76
```

Figure [9] : snipt of Ridge regression model

The Ridge regression model is employed to predict 'levelCancerAntigen' on the test dataset as shown in figure [9]. The model is initialized with an optimal alpha value, best_alpha2, for regularization. then, it is trained on the training data to establish relationships between the selected features and the target variable. The model's predictive performance is assessed through predictions (y_pred_new2) generated on the test data (X_test3). The optimal alphaa value for the Ridge model is reported as 1e-05, a parameter that enhance the extent of regularization and feature selection. also, the Ridge regression model shows high prediction accuracy, evidenced by a low Mean Squared Error (MSE) of 0.38. Moreover, it effectively explains a significant portion of the variance in 'levelCancerAntigen,' achieving an R-squared (R2) score of approximately 0.76. In summary, the Ridge regression model showcases robust predictive capabilities, delivering precise predictions with minimal error and offering valuable insights into the variance of the 'levelCancerAntigen' target variable.

Regression Models' comparison

Regression Model	**Test Error (MSE)**	**R2 Score/std Error**
Least Squares	0.347	0.138
Lasso	0.59	0.62
Ridge	0.38	0.76

From the table above it can be seen that Ridge Regression stands out as the best model for predicting the level Cancer Antigen. It achieves a Root Mean Squared Error (RMSE) of 0.38, indicating highly accurate predictions, and an impressive R-squared value of 0.76, that shiws its ability to explain a significant portion of the variance in 'levelCancerAntigen.'

Also, when it comes to the feature importance using Ridge Regression, the following clinical measures emerge as the most influencing in predicting cancer antigen: 'logCancerVol' ,'logCancerWeight' ,''svi' ' 'logCP,' and 'age.' These measures are the key drivers in cancer prediction and hold critical clinical significance.

Task 2: Classification

Data preparation and preprocessing

```
In [52]: train_directory = pathlib.Path("Dogs_Cats_Data/TRAIN")
         test_directory = pathlib.Path("Dogs_Cats_Data/TEST")
```

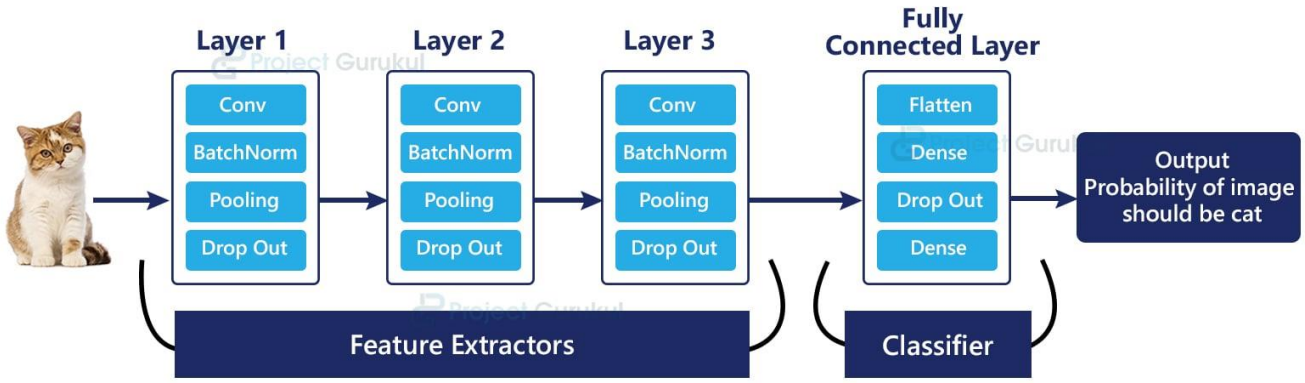
```
In [53]: class_names = os.listdir(train_directory)
         categories = []
         for filename in class_names:
             category = filename.split('.')[0]
             if category == 'dog':
                 categories.append(1)
             else:
                 categories.append(0)

         df = pd.DataFrame({
             'filename': class_names,
             'category': categories
         })
```

Figure [1] : Loading and Preparing the data

In Figure[1], preparing the dataset for the dog and cat classification task by applying the necessary data preprocessing techniques to ensure that the data were ready for training and model analysis the main step is to provide the class score for each image in our dataset by extracting category information from the filenames which could be cat or dog to make this data available to machine learning models by converting these text class letters into equivalent numbers especially using 0 for dog and 1 for dog this encoding is fundamental, because it provides a consistent and standard way for the model to distinguish between the two classes to further structure the dataset it is essential to create a position structure using pandas data frame this data frame contains two columns filename and part the filename column stores the names of the image files while the category column contains the corresponding numeric characters this structured strategy increases the manageability and accessibility of data making it easier to work with it during modeling analysis and solving any potential problems, so this approach increases the consistency and readability of the data simplifies the interpretation of individual data points and ensures that the model is trained on accurate and consistent data is done this data preprocessing step is considered as standard practices in image classification tasks and is an important foundation for building and training effective machine learning models in our field

Building the Model



Figure[2] : The process of CNN model [1]

From Figure [2] , the process of CNN model can be summarized in the following steps [2] :

- Input Layer: This layer serves as the gateway for input image data and transforms it into a one-dimensional array. For instance, if the image is 64x64, resulting in 4096 pixels, this layer will convert it into an array with dimensions (4096, 1).
- Convolutional Layer: This layer is responsible for identifying and extracting key features from the input image.
- Pooling Layer: This layer plays a role in reducing the spatial dimensions of the input image after the convolution process.
- Fully Connected Layer: This layer establishes connections between different layers in the network, facilitating information flow.
- Output Layer: This layer provides the final predicted values, representing the network's output.

```
In [10]: from keras.models import Sequential
         from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

         model = Sequential()

         model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))
         model.add(Dropout(0.25))

         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))
         model.add(Dropout(0.25))

         model.add(Conv2D(128, (3, 3), activation='relu'))
         model.add(BatchNormalization())
         model.add(MaxPooling2D(pool_size=(2, 2)))
         model.add(Dropout(0.25))

         model.add(Flatten())
         model.add(Dense(512, activation='relu'))
         model.add(BatchNormalization())
         model.add(Dropout(0.5))
         model.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes

         model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

         model.summary()

         Model: "sequential"
```

Figure[3] : The code snippet of CNN model

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
batch_normalization (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
dropout (Dropout)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_1 (Dropout)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_2 (Dropout)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12845568
batch_normalization_3 (BatchNormalization)	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1026

Total params: 12942786 (49.37 MB)
Trainable params: 12941314 (49.37 MB)
Non-trainable params: 1472 (5.75 KB)

Figure[4] : CNN model summary

Effective CNN Architecture: The model architecture plays a importannt role in creating an efficient image classifier for cat and dog categorization. the CNN architecture is thoughtfully designed with three Conv2D layers that progressively capture and process image features. It starts with 32 filters in the initial layer and increases to 64 and 128 filters in deeper layers, allowing the network to learn features of varying complexity, critical for accurate classification.

Feature Enhancement: To enhance feature learning, by useing the ReLU activation function after each convolutional layer, a common choice in image classification. Max-pooling layers downscale feature maps, reducing complexity while preserving vital information.

Overfitting Prevention: Combatting overfitting is crucial. incorporating dropout layers with a 0.25 dropout rate after pooling layers, enhancing the model's ability to generalize from training data to unseen images.

Transition to Fully Connected Layers: A flatten layer prepares the data for transition to fully connected layers [3], converting 2D feature maps into a 1D vector.

Dense Layers and Classification: The model includes two dense layers. The first, with 512 neurons, extracts high-level features, while the final dense layer has two neurons, employing a softmax activation function. These neurons represent 'cat' and 'dog,' ensuring the model outputs class probabilities that sum to 1.

Optimized Training: Selecting the right loss function, optimizer, and evaluation metric is critical. Using categorical cross-entropy for loss, RMSprop for optimization, and accuracy for evaluation, aligning with our image classification task.

CNN Appropriateness: For image classification, a Convolutional Neural Network (CNN) is ideal. It automatically learns hierarchical features from visual data, matching the dataset's structured nature. Convolutional layers extract relevant features, pooling layers enhance efficiency, and dropout layers contribute to a robust model.

Data Augmentation for Training

Traning Generator

```
In [29]: train_datagen = ImageDataGenerator(
             rotation_range=15,
             rescale=1./255,
             shear_range=0.1,
             zoom_range=0.2,
             horizontal_flip=True,
             width_shift_range=0.1,
             height_shift_range=0.1
         )

         train_generator = train_datagen.flow_from_dataframe(
             train_df,
             "Dogs_Cats_Data/TRAIN/",
             x_col='filename',
             y_col='category',
             target_size=IMAGE_SIZE,
             class_mode='categorical',
             batch_size=batch_size
         )

         Found 1601 validated image filenames belonging to 2 classes.
```

Figure[5] : Train generator

The train generator prepares training data as shown from figure[5] that by loading images from the specified directory and resizing them to a uniform size. It also transforms class labels into a format compatible with the model's requirements [3].

- **Rotation:** Slight image rotations, up to 15 degrees, allow the model to handle objects from different angles.
- **Rescaling:** Pixel values are rescaled to a standardized range (1/255), ensuring consistent data input.
- **Shearing:** A subtle shear effect is applied, aiding the model in comprehending distorted shapes.
- **Zooming:** Images are zoomed in and out within a 20% range, diversifying the model's ability to recognize objects at various sizes.
- **Horizontal Flip:** Images are mirrored horizontally, exposing the model to different perspectives.
- **Width and Height Shift:** Slight shifts simulate object movements, enhancing the model's generalization.

Generator validation

```
In [30]: validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_dataframe(
    validate_df,
    "Dogs_Cats_Data/TRAIN/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

Figure[6] : Validation Data Generator

The validation Data Generator as shown in figure [6] prepares the validation dataset. Images are rescaled to a range between 0 and 1 for consistency. This generator is configured with the validation dataset's file paths and corresponding labels ('cat' or 'dog'). Images are resized to a specified target size, and they are categorized as 'categorical' data. Additionally, the batch size for processing is set.

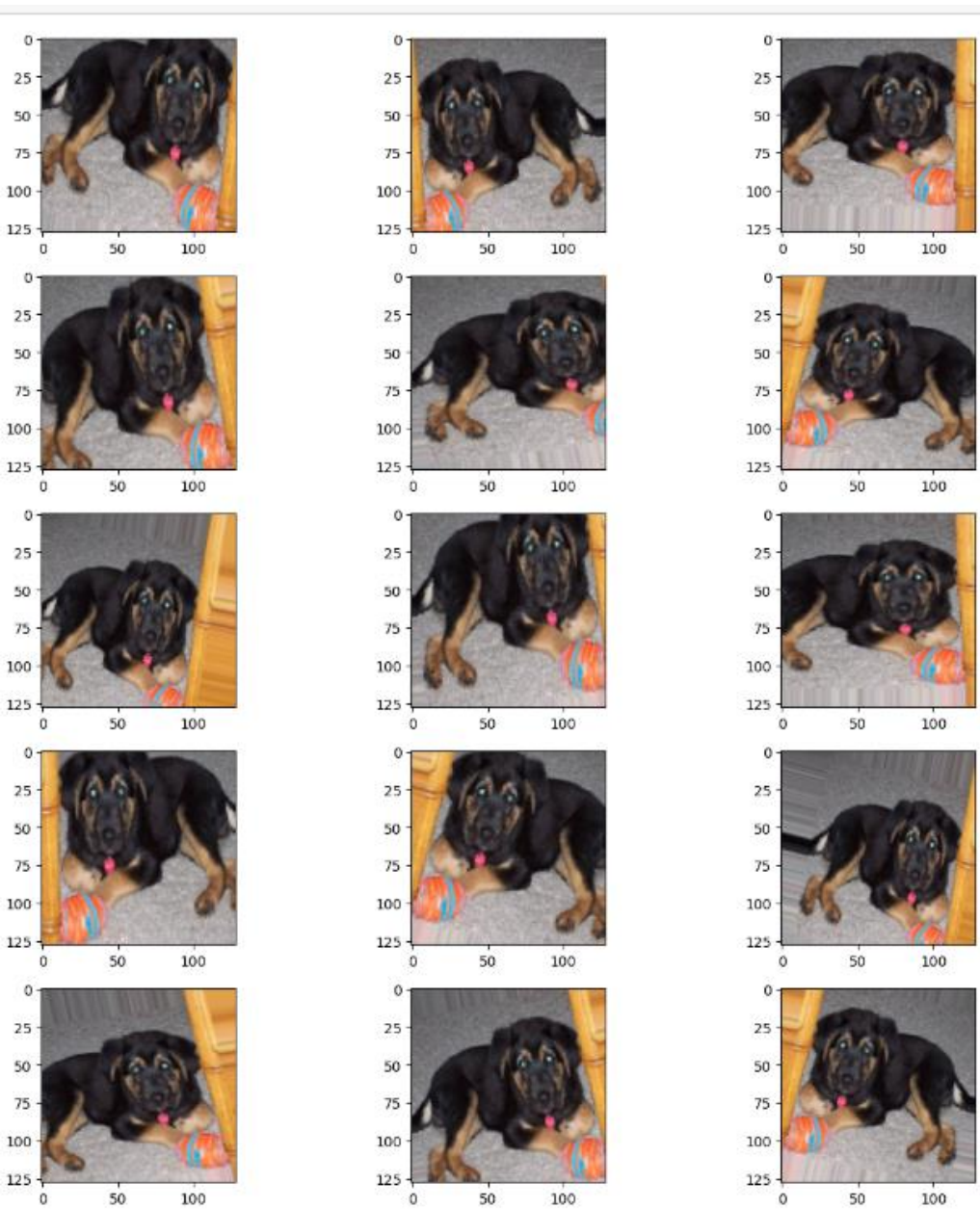
check on generator works

```
In [31]: example_df = train_df.sample(n=1).reset_index(drop=True)
example_generator = train_datagen.flow_from_dataframe(
    example_df,
    "Dogs_Cats_Data/TRAIN/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical'
)
```

```
Found 1 validated image filenames belonging to 1 classes.
```

Figure[7] : Checking Validation Data Generator

This generator is for creating sample data from the training dataset as It selects a random sample of one image and sets up the generator similarly to the validation data generator as it is implemented in figure [7] and the results shown in Figure [8].



Figure[8] : Validation Data Generator example

Fit Model :

Fit model

```
In [33]: epochs=3 if FAST_RUN else 50
         history = model.fit_generator(
             train_generator,
             epochs=epochs,
             validation_data=validation_generator,
             validation_steps=total_validate//batch_size,
             steps_per_epoch=total_train//batch_size,
             callbacks=callbacks
         )
```

```
106/106 [==============================] - ETA: 0s - loss: 0.4715 - accuracy: 0.7755WARNING:tensorflow:Learning rate reductio
n is conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
106/106 [==============================] - 72s 682ms/step - loss: 0.4715 - accuracy: 0.7755 - val_loss: 0.5027 - val_accurac
y: 0.7462 - lr: 0.0010
Epoch 26/50
106/106 [==============================] - ETA: 0s - loss: 0.4712 - accuracy: 0.7711WARNING:tensorflow:Learning rate reductio
n is conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
106/106 [==============================] - 75s 702ms/step - loss: 0.4712 - accuracy: 0.7711 - val_loss: 0.4920 - val_accurac
y: 0.7487 - lr: 0.0010
Epoch 27/50
106/106 [==============================] - ETA: 0s - loss: 0.4510 - accuracy: 0.7926WARNING:tensorflow:Learning rate reductio
n is conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
106/106 [==============================] - 74s 696ms/step - loss: 0.4510 - accuracy: 0.7926 - val_loss: 0.5069 - val_accurac
y: 0.7590 - lr: 0.0010
Epoch 28/50
106/106 [==============================] - ETA: 0s - loss: 0.4637 - accuracy: 0.7825 WARNING:tensorflow:Learning rate reducti
on is conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
106/106 [==============================] - 1391s 13s/step - loss: 0.4637 - accuracy: 0.7825 - val_loss: 0.6484 - val_accurac
y: 0.7333 - lr: 0.0010
```

Figure[9] : Model training

The first step in model training is Setting the Number of Epochs as it shown in figure [9] , The variable epochs is determined based on a conditional statement[2][3]. If FAST_RUN is true, it is set to 3; otherwise, it is set to 50. This allows for flexible training durations, with a shorter training period when FAST_RUN is enabled.

The Model Training: The model.fit_generator() functtion is used to train the CNN model. It takes the following parameters:

train_generator: This is the data generator for the training dataset, which provides batches of images and their corresponding labels for model training.

epochs: The number of epochs or training iterations to run the model. This value is determined based on the FAST_RUN condition.

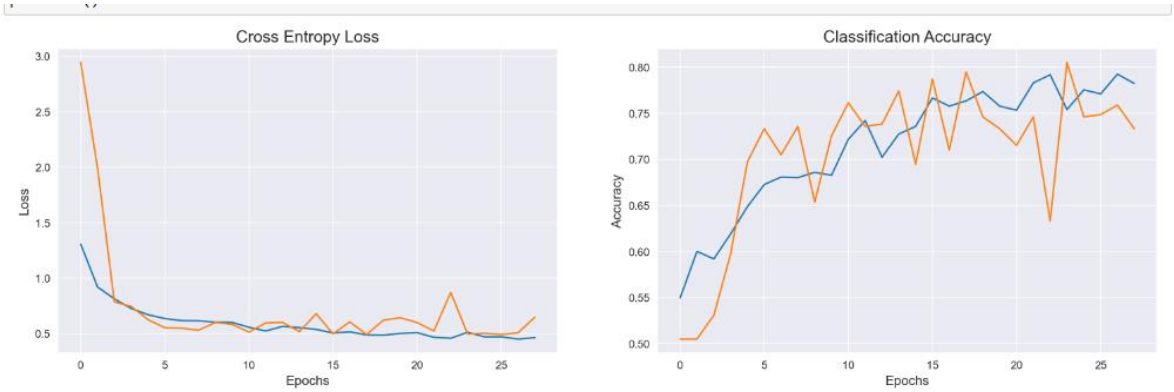
validation_data: The validation data generator, which is used to assess the model's performance on a separate dataset during training.

validation_steps: The number of validation steps to run during each epoch. It's calculated as the total validation data size divided by the batch size.

steps_per_epoch: The number of training steps to run during each epoch. It's calculated as the total training data size divided by the batch size.

callbacks: is applied during training for additional control.

Results



Figure[10] : Plot result

Lastly , this project have effectively constructed a deep neural network model using Convolutional Neural Network (CNN) techniques to achieve a remarkable classification accuracy of 78.25% in discerning between dog and cat images. This model was then employed to make predictions on independent test data, and the results were to assess the accuracy of the model's predictions as shown in figure [10][11].

The Cat vs Dog Image Classification model serves as a testament to the proficient utilization of Convolutional Neural Networks in image classification tasks. By reliably distinguishing between cat and dog images, this project exemplifies the potential of deep learning algorithms in addressing real-world challenges associated with image analysis.

Figure[11] : Visualize Classified Images

References :

[1] Project Gurukul, "Cats vs Dogs Classification using Deep Learning," Project Gurukul, [Online]. Available: https://projectgurukul.org/cats-vs-dogs-classification-deep-learning/

[2] "Convolutional Neural Networks (CNNs)," TensorFlow, [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn

[3] T. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," arXiv preprint arXiv:1804.11191, 2018. [Online]. Available: https://arxiv.org/pdf/1804.11191.pdf