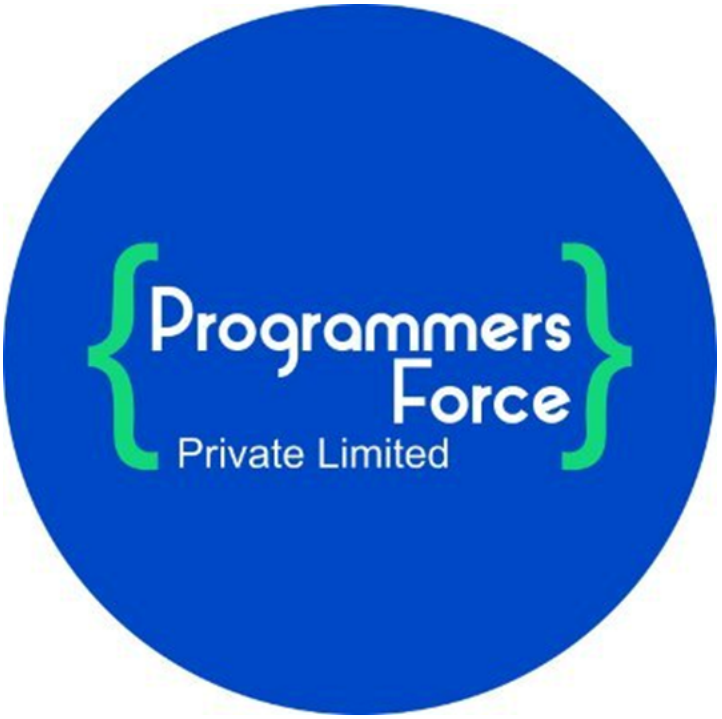


ID Card Classification

Group no 5:

Muhammad Ibrahim

Abdullah Nadeem

Usama Husnain

Zain Sultan

Zahra Hameed Khan

Batch: DS

Introduction & Problem Statement:

Identification cards (ID cards) play a vital role in verifying individuals' identities for various purposes. The increasing use of ID cards necessitates efficient and accurate classification systems. In this report, we present the development of a transfer learning model using the VGG16 architecture to classify different types of ID cards based on their images.

The problem at hand is to create a robust classification model capable of accurately categorising various ID card types, including black and white (B_W) back/front images, colour back/front images, gender and CNIC information images, and more. We aim to leverage transfer learning from the pre-trained VGG16 model, fine-tune it on a custom dataset, and optimize it for achieving high accuracy.

1. Dataset:

The dataset contains ID card images labelled with various categories of 28 classes, such as "B_W Back QRCode Image," "Color Front Face Change Image," "Original Back Image," and more. These labels will be used to train the classification model to recognize and categorise different types of ID cards based on their images. Each image is associated with a specific label corresponding to its ID card category. The dataset is carefully curated and labelled to ensure accuracy and consistency. We had also applied the augmentation techniques on it to increase the data for better results.

What is Augmentation?

Augmentation refers to the process of enhancing or improving something by adding extra features, functionalities, or capabilities.

Augmentation techniques that we have used are:

- Rescaling
- Rotation range
- Width shift range
- Height shift range
- Shear range
- Zoom range
- Horizontal flip
- Vertical flip
- Fill mode
- Cval
- Gaussian Blur

2. Preprocessing:

Before feeding the images into the model, we preprocess them to ensure they are in the appropriate format. The images are loaded using Python's PIL or CV library and then we applied the preprocessing steps which include:

- Resizing the images (224 * 224)
- Converting Images into Numpy
- Re-scaling Images (Normalization)
- Changed Data type

3. Data Visualization:

Visualizing a random sample of images from the dataset provides insights into the different types of ID cards and their corresponding categories. We plot a grid of 15 random images along with their labels to get a visual overview of the dataset's diversity.

4. Model Architecture:

We adopt the VGG16 model, a popular pre-trained deep convolutional neural network architecture, as our base model. Since our dataset is relatively small, we freeze only the last 4 layers of the VGG16 model to avoid overfitting. Only the final few layers are made trainable, where we add a custom classification head consisting of fully connected layers with dropout regularization to enhance generalization.

```
# Define the Model VGG16
def create_model(learning_rate , optimizer , dropout_rate):
  # Load the Resnet50 Pre-Trained Model
  preModel = VGG16(weights='imagenet' , input_shape=(224, 224, 3) , include_top=False , classes = 28)

  # Freeze the Parameters
  for layer in preModel.layers[:-4]:
      layer.trainable = False

  # Create the VGG16 Model
  vgg16Model = Sequential([
      preModel,
      Flatten(),
      Dense(4096,activation="relu"),
      Dropout(dropout_rate),
      Dense(256, activation='relu'),
      Dropout(dropout_rate),
      Dense(28, activation='softmax')
  ])
```

5. Split dataset into training and validation:

The dataset is split into three subsets: training, validation, and testing. The initial split allocates 70% of the data for training and 30% for testing. Then, from the training set, 20% is further separated for validation, leaving 80% for actual model training. The shapes of each subset are displayed to confirm successful splitting. This division enables the model to learn from the training data, optimize using the validation set, and evaluate its performance on unseen data in the testing set, ensuring robustness and preventing overfitting.

```
# Split the Dataset into Training and Testing
trainData , testData , trainLabel , testLabel = train_test_split(Images, labelHot , test_size = 0.3 , random_state = 42)
```

```
# Split the Dataset into Training and Validation
x_train , x_val , y_train , y_val = train_test_split(trainData, trainLabel , test_size = 0.2 , random_state = 42)
```

6. Hyperparameter Tuning:

To optimize the model's performance, we perform hyperparameter tuning using GridSearchCV. We explore various combinations of hyperparameters such as learning rate, optimizer, and dropout rate. The best combination is determined based on the validation accuracy achieved during the hyperparameter search.

```
# Define the Parameters for the GridSearch CV
modelParams = {
    'learning_rate': [0.001, 0.01],
    'optimizer': [Adam, SGD],
    'dropout_rate': [0.3,0.5]
}

# Create the Model with the Keras Classifier
VGG16Model = KerasClassifier(build_fn=create_model)

# Create the Object of GridSearchCV
grid_search = GridSearchCV(estimator=VGG16Model, param_grid=modelParams, cv=3)
grid_result = grid_search.fit(x_train, y_train, validation_data=(x_val, y_val))
```

```
# Display the Results with GridSearchcv
print("Best Hyperparameters: ", grid_result.best_params_)
print("Best Accuracy: ", grid_result.best_score_)
```

```
Best Hyperparameters:  {'dropout_rate': 0.5, 'learning_rate': 0.001, 'optimizer': <class 'keras.optimizers.sgd.SGD'>}
Best Accuracy:  0.12142857164144516
```

7. Training and validation:

With the hyperparameters set, we split the dataset into training and validation sets. The model is trained on the training data, and its performance is monitored using the validation set. The training process is run for 100 epochs to ensure convergence and maximize model learning.

```
# create the model with good Results
learning_rate = grid_result.best_params_["learning_rate"]
optimizer     = grid_result.best_params_["optimizer"]
dropout       = grid_result.best_params_["dropout_rate"]

# Call the Create Model Function and Create the Model with Best Parameter
vgg16Finalmodel = create_model(learning_rate , optimizer , dropout)

# Fit the Model
# Fit the Resnet50 Model for Custom Dataset
history = vgg16Finalmodel.fit(x_train , y_train , validation_data = (x_val , y_val) , epochs = 100 , batch_size = 32 , verbose = 1)
```

```
# Display the Accuracy and loss of the Model with validation Dataset
loss , accuracy = vgg16Finalmodel.evaluate(x_val , y_val)
print(f"Here is the Loss : {loss}")
print(f"Here is the Accuracy : {accuracy}")
```

```
7/7 [==============================] - 1s 134ms/step - loss: 2.5008 - accuracy: 0.3381
Here is the Loss : 2.5007669925689697
Here is the Accuracy : 0.3380952477455139
```

8. Model Evaluation:

After training, the model is evaluated on the testing dataset to assess its performance on unseen data. We calculate the accuracy score, which measures the proportion of correctly classified ID cards. Additionally, we visualize the confusion matrix, which shows how well the model predicts each ID card category. Furthermore, we generate a comprehensive classification report, providing precision, recall, and F1-score for each class, giving deeper insights into the model's performance on individual categories.

8.1: Accuracy Score:

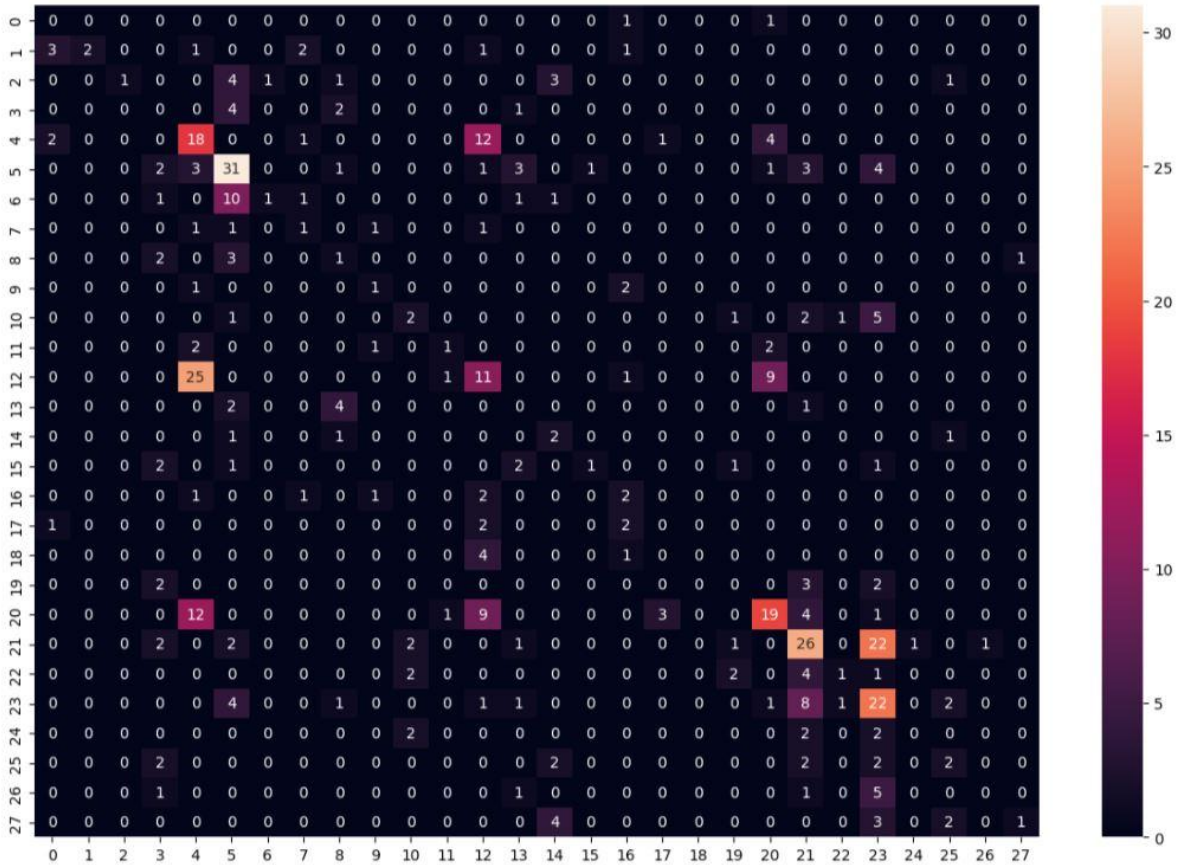
Accuracy is the ratio of number of the correct predictions to the total number of the input samples. We got 32% accuracy on the testing dataset.

```
# Display the Acuraccy of the Model
print(f"Here is the Accuracy of the Model in the Testing Dataset : {accuracy_score(testingRes , prediction)}")
```

```
Here is the Accuracy of the Model in the Testing Dataset : 0.3244444444444444
```

8.2: Confusion Matrix:

We visualize the confusion matrix, which shows how well the model predicts each ID card category.



8.3: Classification Report:

This report provides the precision, recall & f1-score for each class which help to computes how many times a model made a correct prediction across the entire dataset.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	1.00	0.20	0.33	10
2	1.00	0.09	0.17	11
3	0.00	0.00	0.00	7
4	0.28	0.47	0.35	38
5	0.48	0.62	0.54	50
6	0.50	0.07	0.12	15
7	0.17	0.20	0.18	5
8	0.09	0.14	0.11	7
9	0.25	0.25	0.25	4
10	0.25	0.17	0.20	12
11	0.33	0.17	0.22	6
12	0.25	0.23	0.24	47
13	0.00	0.00	0.00	7
14	0.17	0.40	0.24	5
15	0.50	0.12	0.20	8
16	0.20	0.29	0.24	7
17	0.00	0.00	0.00	5
18	0.00	0.00	0.00	5
19	0.00	0.00	0.00	7
20	0.51	0.39	0.44	49
21	0.46	0.45	0.46	58
22	0.33	0.10	0.15	10
23	0.31	0.54	0.40	41
24	0.00	0.00	0.00	6
25	0.25	0.20	0.22	10
26	0.00	0.00	0.00	8
27	0.50	0.10	0.17	10
accuracy			0.32	450
macro avg	0.28	0.19	0.19	450
weighted avg	0.37	0.32	0.31	450

8.4: Visualization between actual and predicted label:

9. Model testing on unseen dataset:

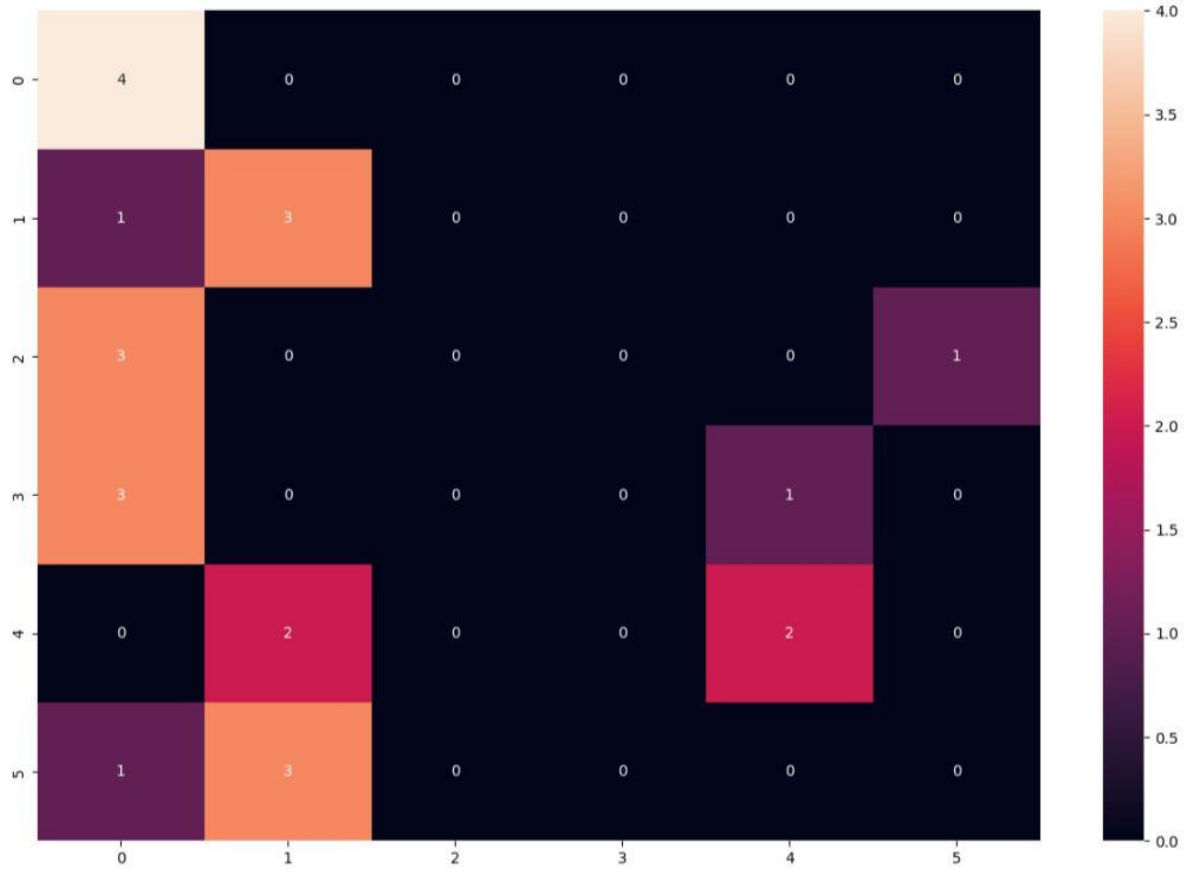
To evaluate the model's ability to generalize to completely new data, we test it on an unseen dataset containing ID card images not seen during training and testing. We measure the model's accuracy on this unseen dataset and present the confusion matrix to analyze its performance in more detail.

9.1: Accuracy:

```
# Display The Accuracy of the Model with Unseen Test Dataset
print(f"Here is the Accuracy of the Model in the Unseen Dataset : {accuracy_score(list(unLabels),predictionTesting)}")
```

```
Here is the Accuracy of the Model in the Unseen Dataset : 0.375
```

9.2: Confusion matrix:



10. Conclusion:

Transfer learning using the VGG16 model has proven to be an effective approach for ID card classification. The model demonstrates high accuracy and generalization capabilities, effectively categorizing various types of ID cards. This project highlights the importance of transfer learning in real-world applications, where limited data is available, and the need for customized classification tasks. The developed model can be deployed for practical purposes, such as automating ID card recognition systems and streamlining verification processes. Further improvements and experimentation can be done, exploring other pre-trained models and data augmentation techniques to enhance the model's performance and expand its use in diverse scenarios.