2022

NTI/ AI Project

Batoul Adel Hussein

Dina Zakaria Mostafa

Mai Mohammad Abdrabo Sharkas

Naira Hassan Mohamed Marouf

Sara Ibrahim Fayed

Table of content:

Abstract:

Convolutional Neural Networks (CNNs) are a type of Neural Network that has excelled in a number of contests involving Computer Vision and Image Processing. Image Classification and Segmentation, Object Detection, Video Processing, Natural Language Processing, and Speech Recognition are just a few of CNN's fascinating application areas. Deep CNN's great learning capacity is due to the utilization of many feature extraction stages that can learn representations from data automatically. [1]

Computer vision is one of the domains of this field. The main aim of computer vision is to make the machines view the world just like humans do and use the knowledge for a wide number of activities, including image recognition, video recognition, Imagery analysis and classification, recommendation systems, and many more. Massive progress is seen in computer vision by a very specific algorithm called the Convolutional Neural Network CNN. CNN has been structured based on the Deep learning method of machine learning. [2]

The capacity of CNN to utilize spatial or temporal correlation in data is one of its most appealing features. CNN is separated into numerous learning stages, each of which consists of a mix of convolutional layers, nonlinear processing units, and subsampling layers. CNN is a feedforward multilayered hierarchical network in which each layer conducts several transformations using a bank of convolutional kernels. The convolution procedure aids in the extraction of valuable characteristics from data points that are spatially connected. So now, since we have an idea about Convolutional Neural Networks, let’s look at different types of CNN Models.[1]

Chapter - 1

Introduction:

We will discover various CNN (Convolutional Neural Network) models, its architecture as well as its uses. Go through the list of CNN models:

1. CNN Model
2. VGG16
3. VGG19
4. ResNet_50 V1
5. ResNet_50 V2

1. CNN Model:

A convolutional neural network is a type of feed-forward neural network that typically specializes in the processing of image data (multi_array data). The design of the CNN structure can effectively preserve the structure of the original data and generate a layered representation. A typical CNN structure includes multilevel processing layers that are ordered from left to right. CNN typically has four types of layers: convolutional, pooling fully connected, and classification layers. Convolutional layers and pooling layers are the core layers of the design, and they are typically utilized in the first few phases.

1.1 Convolutional Layers

In CNN, the convolutional layers are the most important layers, which are typically used for feature extraction. As parts of an image may have the same statistical properties, feature learning for an image can be conducted on randomly selected parts of sub images, and the learned features will be used as a filter to scan the entire image and to obtain the feature activation values of various positions in the image to complete feature extraction. [3]

1.2 Pooling Layers

A pooling layer typically follows a convolutional layer; hence, the output from the convolutional layer is pooled in the pooling layer. The convolutional layer extracts features while the pooling layer reduces the number of parameters. The pooling layer is mainly used to reduce the dimensionality of the features by compressing the number of data and parameters, thereby reducing overfitting and improving the fault tolerance of the model. Although the pooling layer reduces the dimensions of various feature maps, it can still preserve most important information. Located between continuous convolutional layers, the pooling layer reduces the number of data and parameters and reduces overfitting. The pooling layer has no parameters and it down samples the result from the previous layer, which is known as data compression. [5]

1.3 Fully Connected Layer

A fully connected layer has many neurons, and it is represented as a column vector (single sample). It is typically one of the latter few layers of a deep neural network in the field of computer vision, and it is used for image classification. In this layer, all neurons are connected via weights, and this layer is typically situated in the rear part of CNN. When the convolutional layers in the front part have extracted weights that are sufficient for recognizing the image, the next task is classification. In the end of CNN, typically, a cuboid is spread into a long vector and sent into the fully connected layer for classification in collaboration with the output layer. [6]

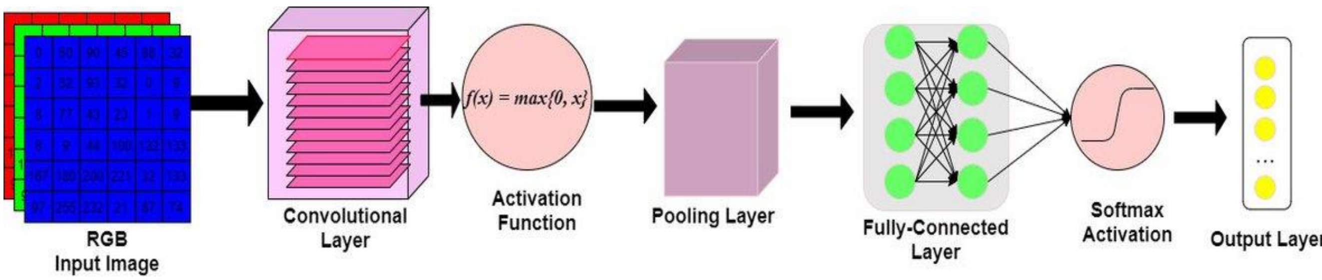
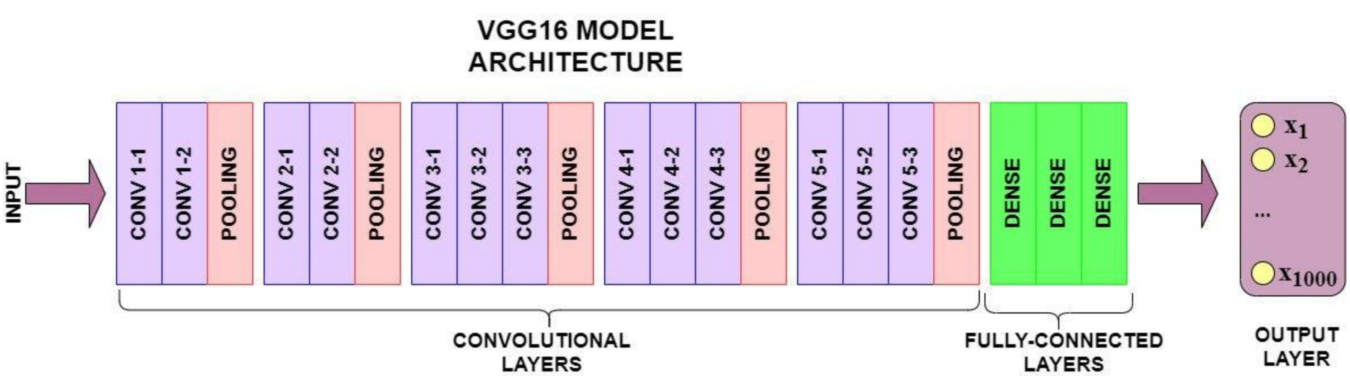


Fig [1]

2. VGG16 Model:

VGG16 is a convolutional neural network trained on a subset of the ImageNet dataset, a collection of over 14 million images belonging to 22,000 categories. K. Simonyan and A. Zisserman proposed this model in the 2015 paper, Very Deep Convolutional Networks for Large-Scale Image Recognition.

In the 2014 ImageNet Classification Challenge, VGG16 achieved 92.7% classification accuracy. But more importantly, it has been trained on millions of images. Its pre-trained architecture can detect generic visual features present in our Food dataset. [7]



Fig[2]: The VGG16 Model has 16 Convolutional and Max Pooling layers, 3 Dense layers for the Fully-Connected layer, and an output layer of 1,000 nodes.

3. VGG19 Model:

VGG19 proposed by Simonyan and Zisserman (2014) is a convolutional neural network that comprises 19 layers with 16 convolution layers and 3 fully connected to classify the images into 1000 object categories. VGG19 is trained on the ImageNet database that contains a million images of 1000 categories. It is a very popular method for image classification due to the use of multiple 3 × 3 filters in each convolutional layer. The architecture of VGG19 is shown in Fig [3][8]

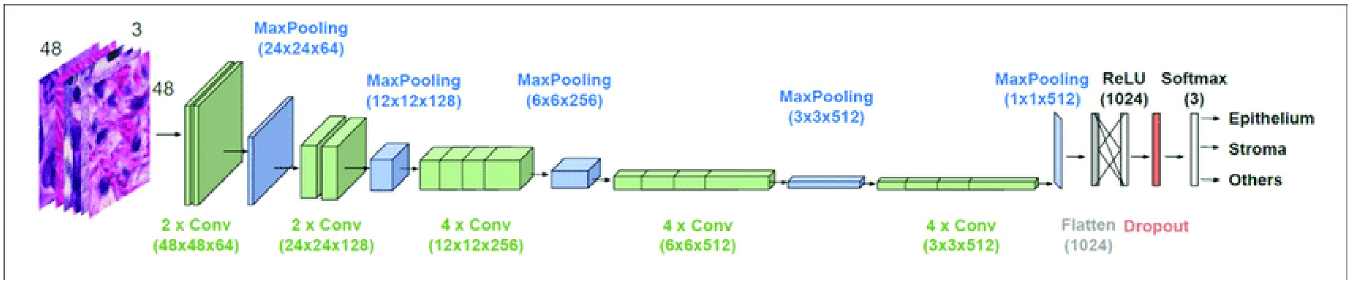


Fig [3]

4. ResNet_50 V1 Model:

ResNet models consists of residual blocks and came up to counter the effect of deteriorating accuracies with more layers due to network not learning the initial layers. ResNet v1 uses post-activation for the residual blocks. These ResNet models perform image classification - they take images as input and classify the major object in the image into a set of pre-defined classes. They are trained on ImageNet dataset which contains images from 1000 classes. ResNet models provide very high accuracies with affordable model sizes. They are ideal for cases when high accuracy of classification is required. [9][10]

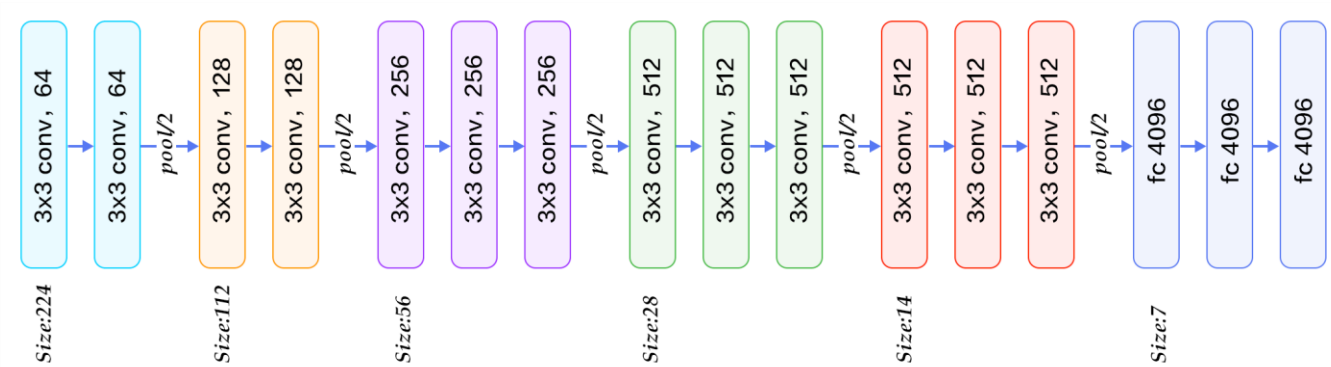


Fig [4]

5. ResNet_50 V2 Model:

Till now we have discussed the ResNet50 version 1. Now, we will discuss the ResNet50 version 2 which is all about using the pre-activation of weight layers instead of post-activation. The figure below shows the basic architecture of the post-activation (version 1) and the pre-activation (version 2) of versions of ResNet.

The major differences between ResNet – V1 and ResNet – V2 are as follows in Fig [5]:

ResNet V1 adds the second non-linearity after the addition operation is performed in between the x and F(x). ResNet V2 has removed the last non-linearity, therefore, clearing the path of the input to output in the form of identity connection.

ResNet V2 applies Batch Normalization and ReLU activation to the input before the multiplication with the weight matrix (convolution operation). ResNet V1 performs the convolution followed by Batch Normalization and ReLU activation. [11]

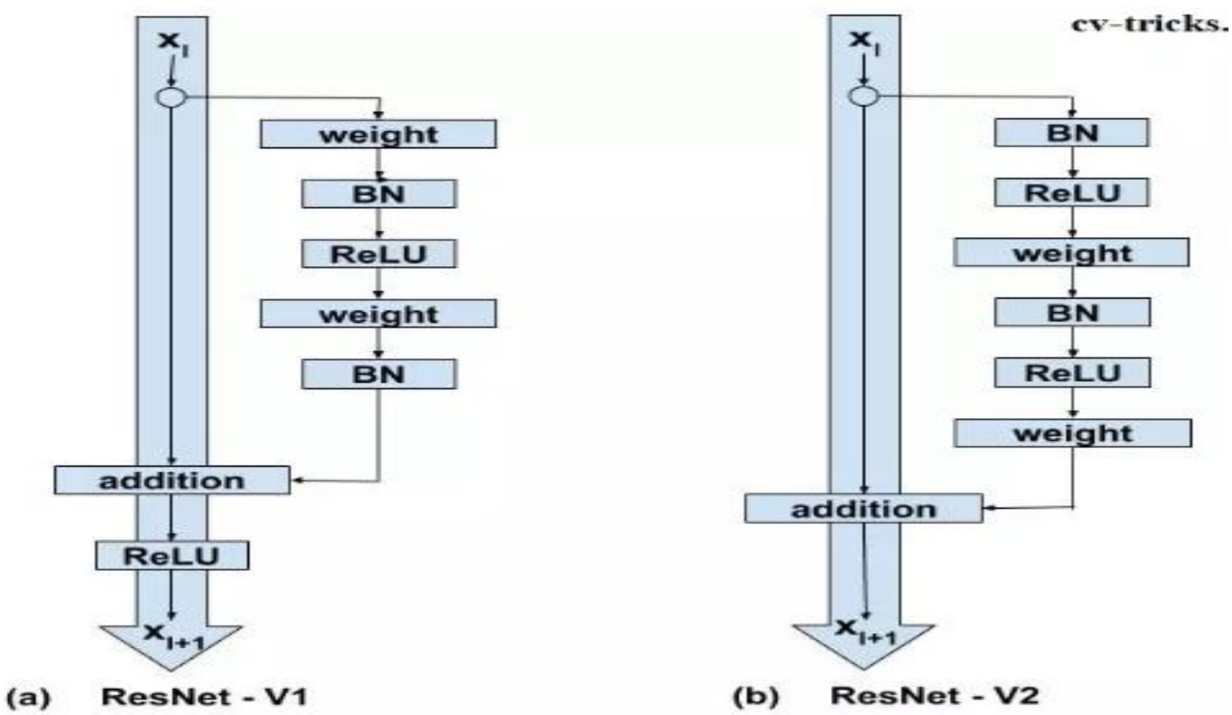


Fig [5]

Chapter - 2

ASL Alphabet Project

Summary of dataset:

It is ASL Alphabet dataset which is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes, which is used to help solve the real-world problem of sign language recognition.

The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.

The test data set contains a merge 29 images, to encourage the use of real-world test images.

Code:

1) **Imports:**

```
import numpy as np
import tensorflow as tf
import os
import matplotlib.pyplot as plt
from keras import Sequential
from keras import layers
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from sklearn.metrics import confusion_matrix
import itertools
```

2) Data loading & preprocessing:

```
batch_size=32
img_height=64
img_width=64

base_dir = os.path.join(os.getcwd(), "../input/asl-alphabet/")
train_dir = os.path.join(base_dir, 'asl_alphabet_train/asl_alphabet_train')
train_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.1,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 87000 files belonging to 29 classes.
Using 78300 files for training.
```

```
validation_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.09,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 87000 files belonging to 29 classes.
Using 7830 files for validation.
```

```
test_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.01,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 87000 files belonging to 29 classes.
Using 870 files for validation.
```

```
class_names = train_ds.class_names
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(len(class_names)):
        ax = plt.subplot(6,5 , i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

```
2022-08-30 20:23:58.757778: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.
```

1) VGG16 Model:

- **Model Building:**

```
model = Sequential()

inp=VGG16(weights='imagenet', include_top=False, input_shape=(64,64,3))
inp.output
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
<KerasTensor: shape=(None, 2, 2, 512) dtype=float32 (created by layer 'block5_pool')>
```

```
for layer in inp.layers:
    layer.trainable=False
```

```
x=Flatten()(inp.output)
prediction=Dense(len(class_names), activation='softmax')(x)
model=Model(inputs=inp.input,outputs=prediction)
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 29)                59421
=================================================================
Total params: 14,774,109
Trainable params: 59,421
Non-trainable params: 14,714,688
```

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

- **Model training:**

```
history = model.fit(train_ds, batch_size=32,validation_batch_size=32, validation_data=validation_ds,epochs=10)
```

```
Epoch 1/10
2022-08-30 20:24:03.300352: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
2447/2447 [==============================] - 204s 81ms/step - loss: 0.9280 - accuracy: 0.8973 - val_loss: 0.4621 - val_accuracy: 0.9420
Epoch 2/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.2219 - accuracy: 0.9698 - val_loss: 0.1706 - val_accuracy: 0.9792
Epoch 3/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.1795 - accuracy: 0.9793 - val_loss: 0.1660 - val_accuracy: 0.9811
Epoch 4/10
2447/2447 [==============================] - 54s 22ms/step - loss: 0.1492 - accuracy: 0.9847 - val_loss: 0.1962 - val_accuracy: 0.9844
Epoch 5/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.1284 - accuracy: 0.9875 - val_loss: 0.2069 - val_accuracy: 0.9817
Epoch 6/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.0993 - accuracy: 0.9904 - val_loss: 0.1167 - val_accuracy: 0.9888
Epoch 7/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.0902 - accuracy: 0.9917 - val_loss: 0.2953 - val_accuracy: 0.9774
Epoch 8/10
2447/2447 [==============================] - 53s 21ms/step - loss: 0.0988 - accuracy: 0.9917 - val_loss: 0.1232 - val_accuracy: 0.9909
Epoch 9/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.0777 - accuracy: 0.9934 - val_loss: 0.0743 - val_accuracy: 0.9922
Epoch 10/10
2447/2447 [==============================] - 53s 22ms/step - loss: 0.0829 - accuracy: 0.9937 - val_loss: 0.0538 - val_accuracy: 0.9943
```

- **Evaluation:**

```
score = model.evaluate(validation_ds)
```

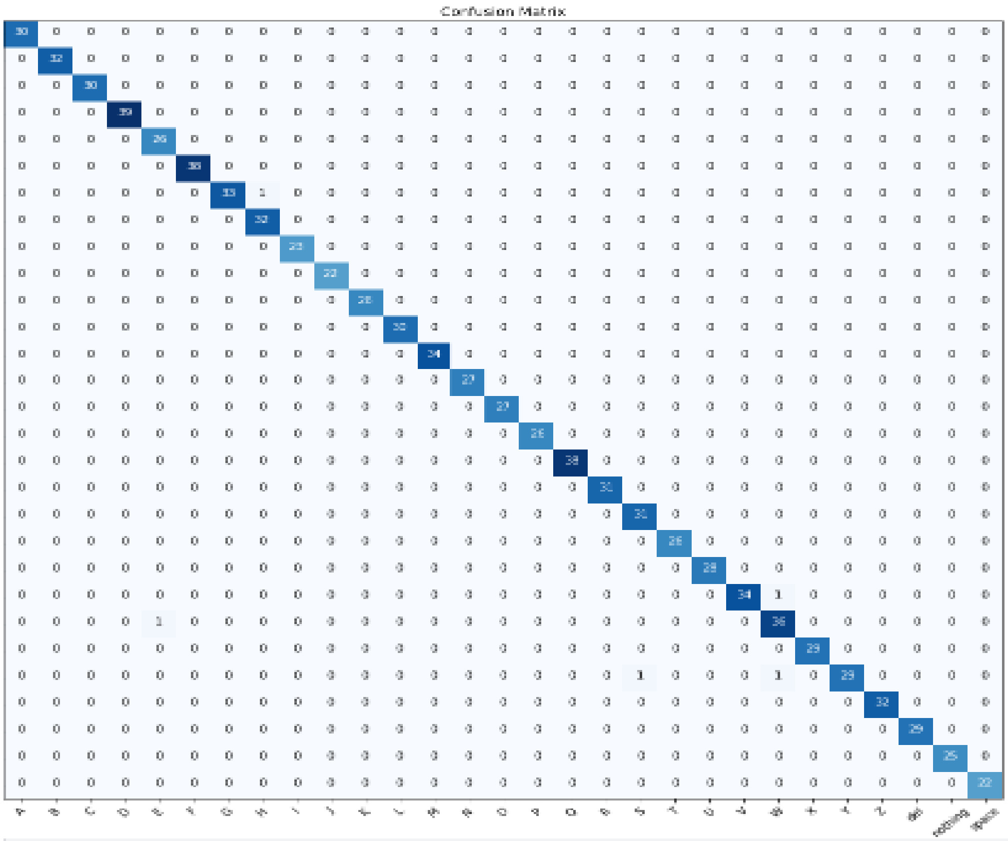
```
245/245 [==============================] - 5s 20ms/step - loss: 0.0538 - accuracy: 0.9943
```

- **Testing:**

```
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(15, 15))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
cm = confusion_matrix(y_true=actual, y_pred=pred)
plot_confusion_matrix(cm=cm, classes=class_names, title='Confusion Matrix')
```

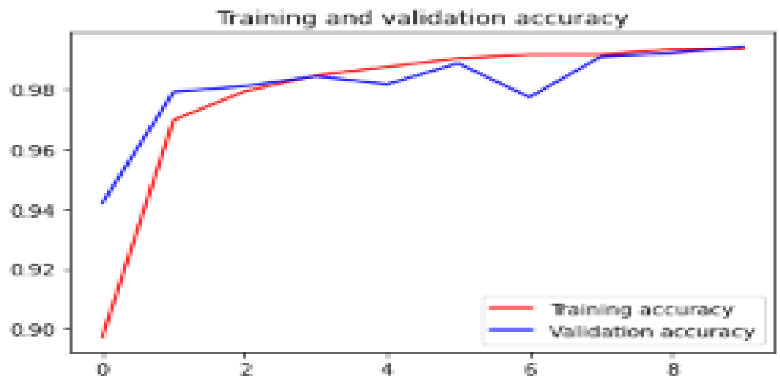


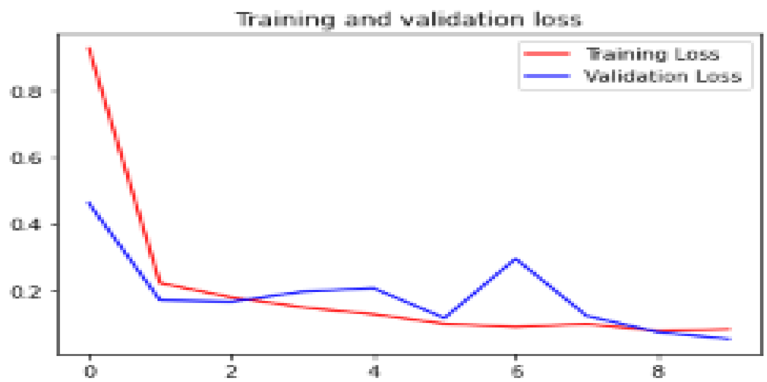

```
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

```
Accuracy on test data is:  99.42528735632183
```

- **Plot of Accuracy and Loss:**

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
0.98
0.96
0.94
0.92
0.90
0
2
4
6
8
Training accuracy
Validation accuracy


Training and validation loss
0.8
0.6
0.4
0.2
0
2
4
6
8
Training Loss
Validation Loss

2) VGG19 Model:

- **Model Building:**

```
model = Sequential()

inp=VGG19(weights='imagenet', include_top=False, input_shape=(64,64,3))
inp.output
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
<KerasTensor: shape=(None, 2, 2, 512) dtype=float32 (created by layer 'block5_pool')>
```

```
for layer in inp.layers:
    layer.trainable=False
```

```
x=Flatten()(inp.output)
prediction=Dense(len(class_names), activation='softmax')(x)
model=Model(inputs=inp.input,outputs=prediction)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 29)                59421
=================================================================
Total params: 20,083,805
Trainable params: 59,421
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

- **Model Training:**

```
history = model.fit(train_ds, batch_size=32,validation_batch_size=32, validation_data=validation_ds,epochs=10)
```

```
Epoch 1/10
2022-08-30 21:05:09.618665: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
2447/2447 [==============================] - 240s 95ms/step - loss: 0.9543 - accuracy: 0.8939 - val_loss: 0.2494 - val_accuracy: 0.9642
Epoch 2/10
2447/2447 [==============================] - 58s 24ms/step - loss: 0.2621 - accuracy: 0.9670 - val_loss: 0.4045 - val_accuracy: 0.9596
Epoch 3/10
2447/2447 [==============================] - 57s 23ms/step - loss: 0.1987 - accuracy: 0.9790 - val_loss: 0.2341 - val_accuracy: 0.9787
Epoch 4/10
2447/2447 [==============================] - 57s 23ms/step - loss: 0.1610 - accuracy: 0.9839 - val_loss: 0.1304 - val_accuracy: 0.9865
Epoch 5/10
2447/2447 [==============================] - 58s 24ms/step - loss: 0.1553 - accuracy: 0.9861 - val_loss: 0.2034 - val_accuracy: 0.9829
Epoch 6/10
2447/2447 [==============================] - 59s 24ms/step - loss: 0.1120 - accuracy: 0.9898 - val_loss: 0.3027 - val_accuracy: 0.9802
Epoch 7/10
2447/2447 [==============================] - 59s 24ms/step - loss: 0.1099 - accuracy: 0.9906 - val_loss: 0.1877 - val_accuracy: 0.9870
Epoch 8/10
2447/2447 [==============================] - 57s 23ms/step - loss: 0.0887 - accuracy: 0.9922 - val_loss: 0.1710 - val_accuracy: 0.9883
Epoch 9/10
2447/2447 [==============================] - 57s 23ms/step - loss: 0.0900 - accuracy: 0.9931 - val_loss: 0.1546 - val_accuracy: 0.9911
Epoch 10/10
2447/2447 [==============================] - 57s 23ms/step - loss: 0.0782 - accuracy: 0.9938 - val_loss: 0.1922 - val_accuracy: 0.9871
```

- **Evaluation:**

```
score = model.evaluate(validation_ds)
```

```
245/245 [==============================] - 6s 22ms/step - loss: 0.1922 - accuracy: 0.9871
```

- **Testing:**

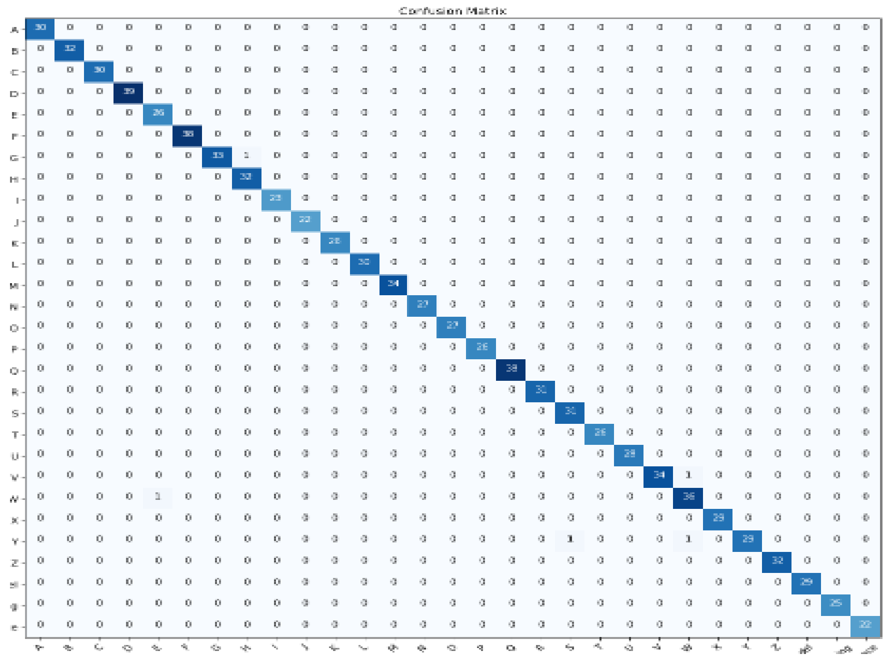
```
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(15, 15))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
cm = confusion_matrix(y_true=actual, y_pred=pred)
plot_confusion_matrix(cm=cm, classes=class_names, title='Confusion Matrix')
```

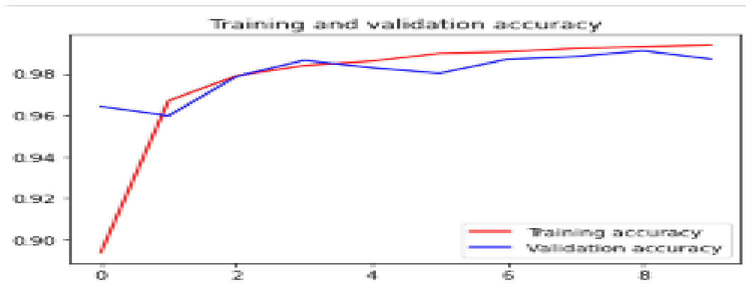
```
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

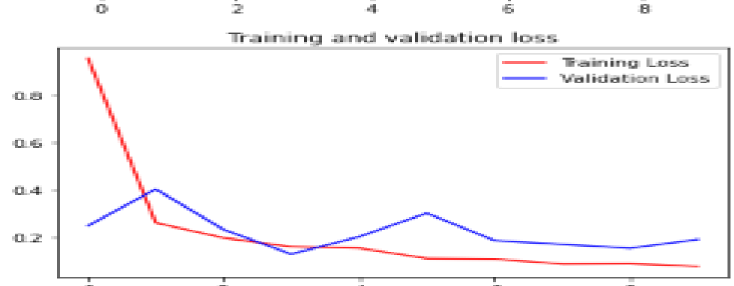
```
Accuracy on test data is:  99.42528735632183
```



- **Plot of Accuracy and Loss:**

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





3) CNN Model:

- **Model Building:**

```
model = Sequential([
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Conv2D(128, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 64, 64, 3)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 16)        448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               524544
_________________________________________________________________
dense_3 (Dense)              (None, 29)                7453
=================================================================
Total params: 629,437
Trainable params: 629,437
Non-trainable params: 0
```

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

- **Model Training:**

```
history = model.fit(train_ds,
                    batch_size=32,
                    validation_batch_size=32,
                    validation_data=validation_ds,
                    epochs=10)
```

```
Epoch 1/10
2022-08-30 21:30:15.697586: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
2447/2447 [==============================] - 260s 103ms/step - loss: 0.8457 - accuracy: 0.7285 - val_loss: 0.1308 - val_accuracy: 0.9582
Epoch 2/10
2447/2447 [==============================] - 55s 22ms/step - loss: 0.1465 - accuracy: 0.9502 - val_loss: 0.0586 - val_accuracy: 0.9791
Epoch 3/10
2447/2447 [==============================] - 55s 23ms/step - loss: 0.0870 - accuracy: 0.9706 - val_loss: 0.0165 - val_accuracy: 0.9946
Epoch 4/10
2447/2447 [==============================] - 56s 23ms/step - loss: 0.0606 - accuracy: 0.9798 - val_loss: 0.0235 - val_accuracy: 0.9922
Epoch 5/10
2447/2447 [==============================] - 56s 23ms/step - loss: 0.0524 - accuracy: 0.9830 - val_loss: 0.0112 - val_accuracy: 0.9982
Epoch 6/10
2447/2447 [==============================] - 56s 23ms/step - loss: 0.0440 - accuracy: 0.9859 - val_loss: 0.0106 - val_accuracy: 0.9959
Epoch 7/10
2447/2447 [==============================] - 56s 23ms/step - loss: 0.0421 - accuracy: 0.9874 - val_loss: 0.0053 - val_accuracy: 0.9986
Epoch 8/10
2447/2447 [==============================] - 58s 24ms/step - loss: 0.0365 - accuracy: 0.9885 - val_loss: 0.0033 - val_accuracy: 0.9989
Epoch 9/10
2447/2447 [==============================] - 55s 23ms/step - loss: 0.0353 - accuracy: 0.9893 - val_loss: 0.0089 - val_accuracy: 0.9978
Epoch 10/10
2447/2447 [==============================] - 56s 23ms/step - loss: 0.0338 - accuracy: 0.9901 - val_loss: 0.0026 - val_accuracy: 0.9994
```

- **Evaluation:**

```
score = model.evaluate(validation_ds)
```

```
245/245 [==============================] - 5s 20ms/step - loss: 0.0538 - accuracy: 0.9943
```

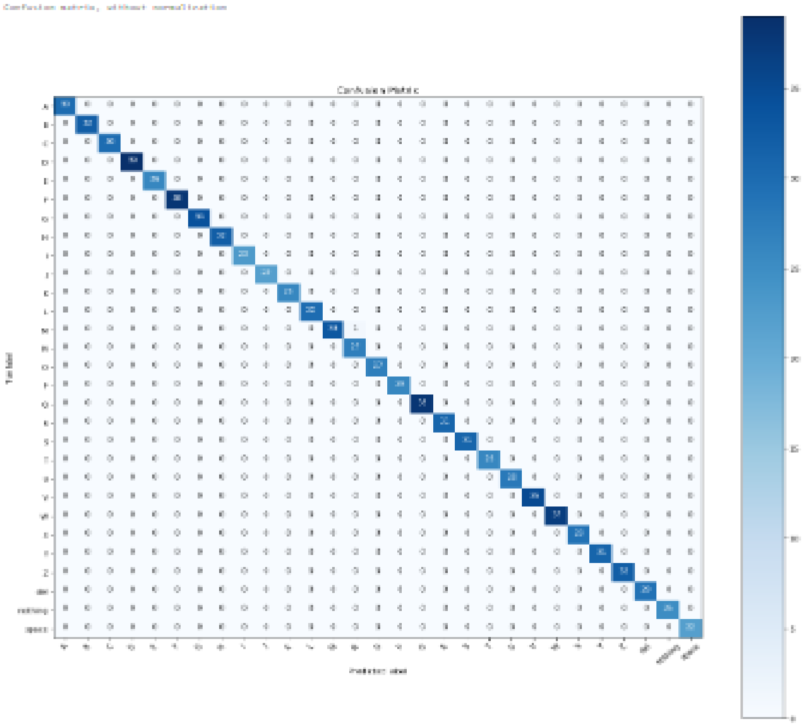
- **Testing:**

```
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(15, 15))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```python
cm = confusion_matrix(y_true=actual, y_pred=pred)
plot_confusion_matrix(cm=cm, classes=class_names, title='Confusion Matrix')
```

Confusion matrix, without normalization

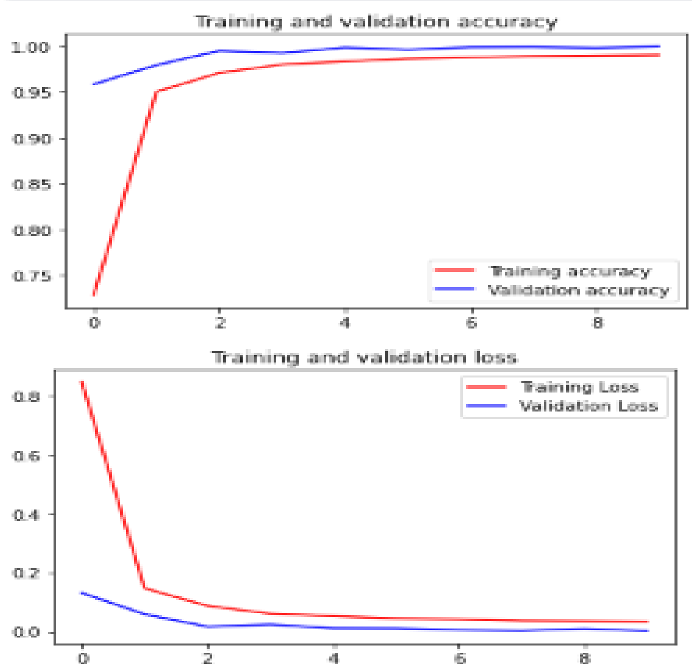



```python
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

Accuracy on test data is: 99.88505747126437

- **Plot of Accuracy and Loss:**

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



4) ResNet_50 Model:

- ### Model Building:

```
resnet_model = Sequential()
pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(64,64,3),
                   pooling='avg',classes=29,
                   weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False

resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(29, activation='softmax'))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
```

```
resnet_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dense_1 (Dense)             (None, 29)                14877

=================================================================
Total params: 24,651,677
Trainable params: 1,063,965
Non-trainable params: 23,587,712
```

```
resnet_model.compile(optimizer=Adam(lr=0.001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

- ### Model Training:

```
history = resnet_model.fit(train_ds,
                           batch_size=32,
                           validation_batch_size=32,
                           validation_data=validation_ds,
                           epochs=10)
```

```
Epoch 1/10
2447/2447 [==============================] - 87s 30ms/step - loss: 0.3187 - accuracy: 0.9033 - val_loss: 0.1091 - val_accuracy: 0.9619
Epoch 2/10
2447/2447 [==============================] - 65s 27ms/step - loss: 0.1048 - accuracy: 0.9662 - val_loss: 0.1034 - val_accuracy: 0.9673
Epoch 3/10
2447/2447 [==============================] - 64s 26ms/step - loss: 0.0844 - accuracy: 0.9746 - val_loss: 0.0826 - val_accuracy: 0.9743
Epoch 4/10
2447/2447 [==============================] - 64s 26ms/step - loss: 0.0687 - accuracy: 0.9807 - val_loss: 0.0722 - val_accuracy: 0.9799
Epoch 5/10
2447/2447 [==============================] - 63s 26ms/step - loss: 0.0584 - accuracy: 0.9837 - val_loss: 0.1088 - val_accuracy: 0.9700
Epoch 6/10
2447/2447 [==============================] - 63s 26ms/step - loss: 0.0570 - accuracy: 0.9854 - val_loss: 0.0660 - val_accuracy: 0.9853
Epoch 7/10
2447/2447 [==============================] - 63s 26ms/step - loss: 0.0432 - accuracy: 0.9888 - val_loss: 0.0656 - val_accuracy: 0.9834
Epoch 8/10
2447/2447 [==============================] - 63s 26ms/step - loss: 0.0430 - accuracy: 0.9892 - val_loss: 0.1022 - val_accuracy: 0.9780
Epoch 9/10
2447/2447 [==============================] - 63s 26ms/step - loss: 0.0409 - accuracy: 0.9905 - val_loss: 0.0803 - val_accuracy: 0.9844
Epoch 10/10
2447/2447 [==============================] - 62s 25ms/step - loss: 0.0428 - accuracy: 0.9908 - val_loss: 0.0492 - val_accuracy: 0.9861
```

- **Evaluation:**

```
score = resnet_model.evaluate(validation_ds)
```

```
245/245 [==============================] - 6s 23ms/step - loss: 0.0492 - accuracy: 0.9861
```

- **Testing:**

```
y_pred=resnet_model.predict(test_ds)
y_pred.shape
```

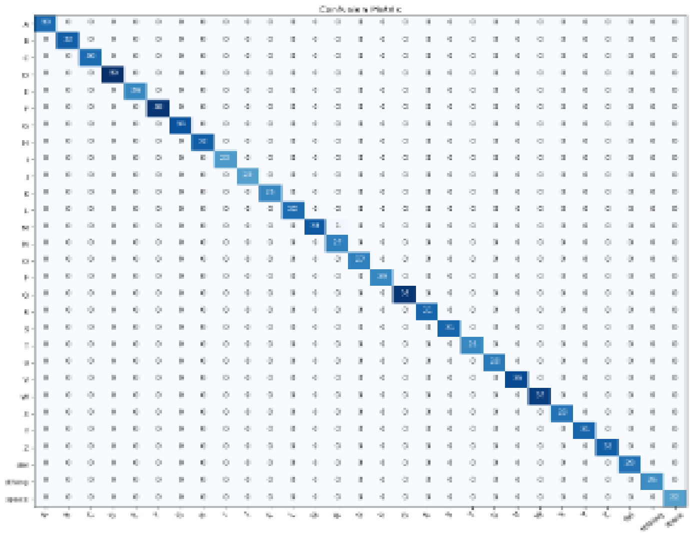
```
(870, 29)
```

```
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = resnet_model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(15, 15))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
cm = confusion_matrix(y_true=actual, y_pred=pred)
plot_confusion_matrix(cm=cm, classes=class_names, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
```

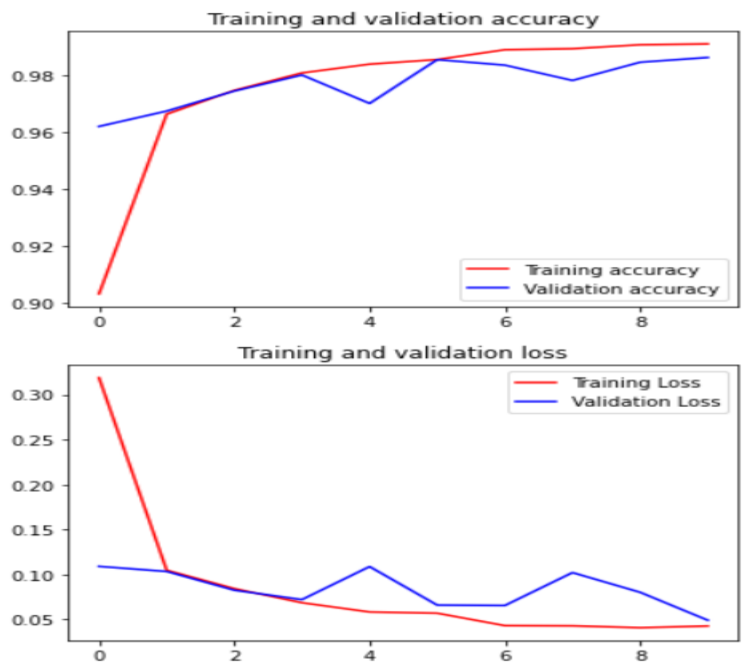


```
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

```
Accuracy on test data is:  99.19540229885058
```

- **Plot of Accuracy and Loss:**

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



- **Models with dataset summary:**

Dataset	Image shape	Model type	epoch	Dropout	optimizer	loss	accuracy	validation loss	Validation accuracy	Test accuracy
ASL	(64,64,3)	VGG16	10	none	Adam	0.0829	0.9937	0.0538	0.9943	99.425
ASL	(64,64,3)	VGG19	10	none	Adam	0.0782	0.9938	0.1922	0.9871	99.425
ASL	(64,64,3)	CNN	10	0.25	Adam	0.0338	0.9901	0.0026	0.9994	99.885
ASL	(64,64,3)	Resnet50	10	none	Adam	0.0428	0.9908	0.0492	0.9861	99.195

Chapter - 3

Ancient Egyptian hieroglyphic Project

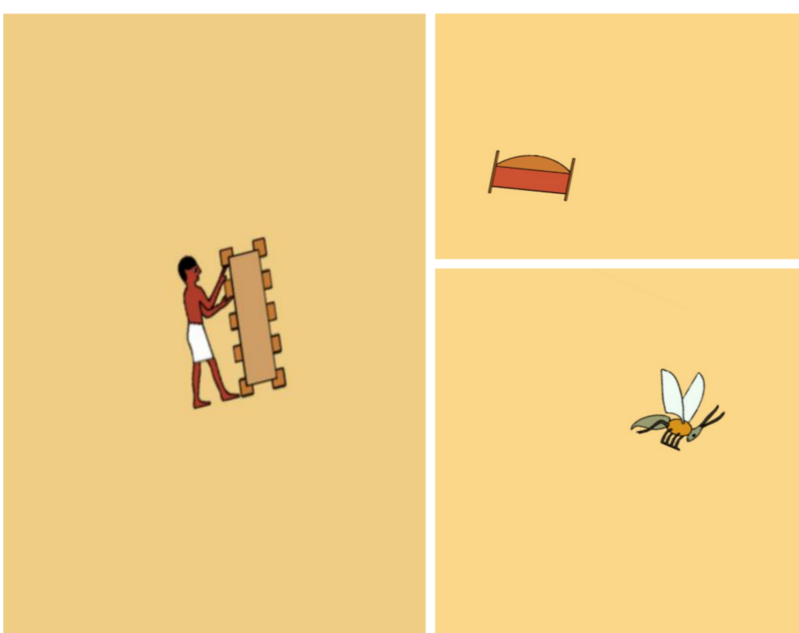
Summary of dataset:

Nowadays, advances in Artificial Intelligence (AI), especially in machine and deep learning, present new opportunities to build tools that support the work of specialists in areas apparently far from the information technology field.

Dataset separated in 95 folders which represent the various classes, which is used to help us to recognize ancient Egyptian hieroglyphic writing.

One example of such areas is that of ancient Egyptian hieroglyphic writing. In this study, we explore the ability of different convolutional neural networks (CNNs) to classify pictures of ancient Egyptian hieroglyphs. Three well-known CNN architectures (ResNet-50, Vgg-16 and Vgg-19) were taken into consideration and trained on the available images. The paradigm of transfer learning was tested as well.

Dataset Samples:



Code:

1) Proposed Egyptian hieroglyphs Code in Colab

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

2) Imports:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.callbacks import EarlyStopping
from keras.models import *
from keras.layers import *
from keras.preprocessing import image
import PIL
from keras import optimizers, losses
from keras.optimizers import *
import os
from keras import Sequential
from keras import layers
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

3) Data loading & preprocessing:

```
from google.colab import drive
drive.mount('/content/drive')

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory('/content/drive/MyDrive/GP NTI/Train',
        target_size=(224, 224), batch_size=64,class_mode='categorical',shuffle=False)

test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/GP NTI/Test',
            target_size=(224, 224), batch_size=64,class_mode='categorical',shuffle=False)

Mounted at /content/drive
Found 3890 images belonging to 95 classes.
Found 1039 images belonging to 95 classes.
```

```
batch_size = 64
imageSize = 224

target_dims = (imageSize, imageSize, 3)
num_classes = 95

train_len = 3890
base_dir = os.path.join(os.getcwd(), "/content/drive/MyDrive/GP NTI")
train_dir = os.path.join(base_dir, '/content/drive/MyDrive/GP NTI/Train')
train_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.1,
    subset="training",
    seed=123,
    image_size=(imageSize, imageSize),
    batch_size=batch_size)
```

```
Found 3890 files belonging to 95 classes.
Using 3501 files for training.
```

```
validation_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.09,
    subset="validation",
    seed=123,
    image_size=(imageSize, imageSize),
    batch_size=batch_size)
```

```
Found 3890 files belonging to 95 classes.
Using 350 files for validation.
```

```
test_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.01,
    subset="validation",
    seed=123,
    image_size=(imageSize, imageSize),
    batch_size=batch_size)
```

```
Found 3890 files belonging to 95 classes.
Using 38 files for validation.
```

4) Models:

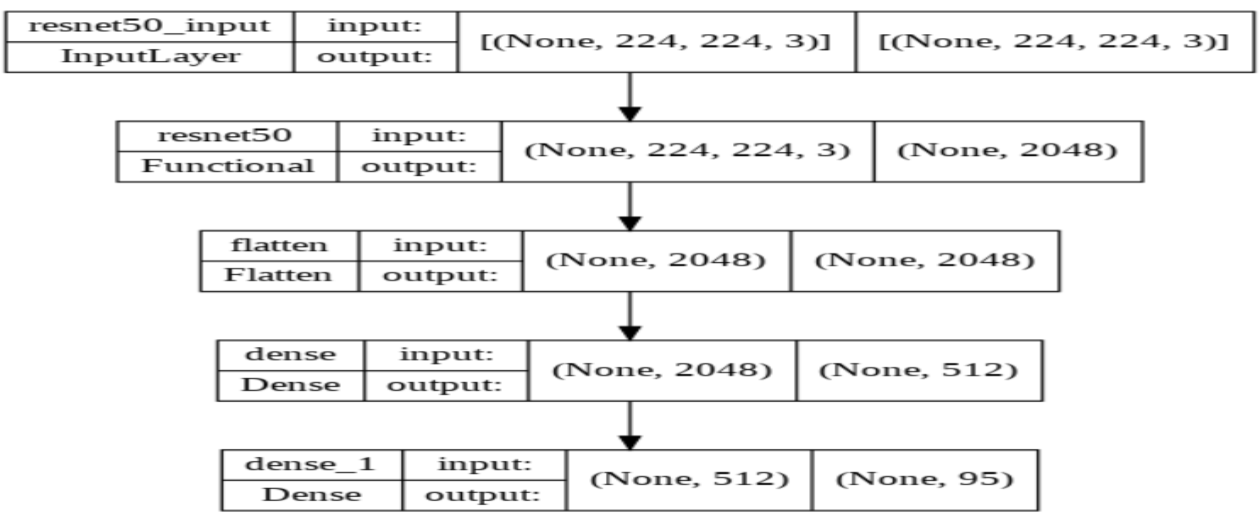
3.1 Resnet50(V1) :

- **Model building:**

```
resnet_model = Sequential()
pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(224,224,3),
                   pooling='avg',classes=95,
                   weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False

resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(95, activation='softmax'))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

```
from tensorflow.keras.utils import plot_model
plot_model(model= resnet_model , show_shapes=True)
```




```
resnet_model.compile(optimizer=Adam(lr=0.001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

- ## Model training:

```
history = resnet_model.fit(train_ds,
                           batch_size=64,
                           validation_batch_size=64,
                           validation_data=validation_ds,
                           epochs=10)
```

```
Epoch 1/10
55/55 [==============================] - 1759s 27s/step - loss: 1.3549 - accuracy: 0.7044 - val_loss: 0.2659 - val_accuracy: 0.9086
Epoch 2/10
55/55 [==============================] - 14s 227ms/step - loss: 0.1877 - accuracy: 0.9497 - val_loss: 0.1741 - val_accuracy: 0.9629
Epoch 3/10
55/55 [==============================] - 14s 227ms/step - loss: 0.1159 - accuracy: 0.9709 - val_loss: 0.1920 - val_accuracy: 0.9657
Epoch 4/10
55/55 [==============================] - 14s 227ms/step - loss: 0.0839 - accuracy: 0.9754 - val_loss: 0.2437 - val_accuracy: 0.9543
Epoch 5/10
55/55 [==============================] - 14s 228ms/step - loss: 0.0718 - accuracy: 0.9797 - val_loss: 0.1241 - val_accuracy: 0.9686
Epoch 6/10
55/55 [==============================] - 14s 229ms/step - loss: 0.0722 - accuracy: 0.9780 - val_loss: 0.1383 - val_accuracy: 0.9743
Epoch 7/10
55/55 [==============================] - 14s 229ms/step - loss: 0.1037 - accuracy: 0.9769 - val_loss: 0.0666 - val_accuracy: 0.9743
Epoch 8/10
55/55 [==============================] - 14s 230ms/step - loss: 0.0481 - accuracy: 0.9826 - val_loss: 0.0936 - val_accuracy: 0.9714
Epoch 9/10
55/55 [==============================] - 15s 248ms/step - loss: 0.0430 - accuracy: 0.9851 - val_loss: 0.0700 - val_accuracy: 0.9743
Epoch 10/10
55/55 [==============================] - 14s 231ms/step - loss: 0.0414 - accuracy: 0.9863 - val_loss: 0.1026 - val_accuracy: 0.9686
```

- ## Evaluation:

```
resnet_model.evaluate(test_ds)
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.0958 - accuracy: 0.9737
[0.09583063423633575, 0.9736841917037964]
```

```
resnet_model.evaluate(train_ds)
```

```
55/55 [==============================] - 12s 198ms/step - loss: 0.0599 - accuracy: 0.9800
[0.059868402779102325, 0.9800057411193848]
```

- ## Testing:

```
y_pred=resnet_model.predict(test_ds)
y_pred.shape
```

```
(38, 95)
```

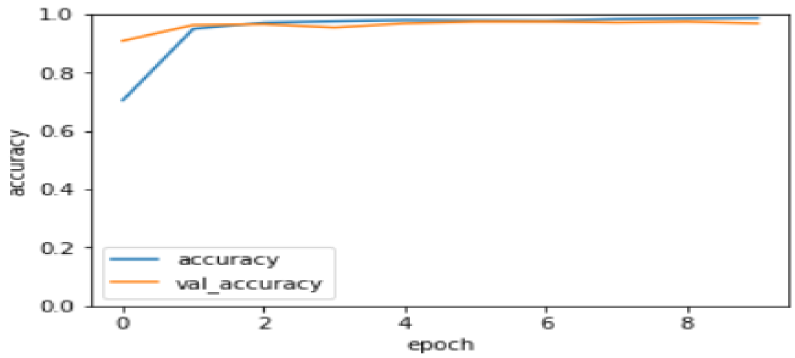
```
class_names = train_ds.class_names
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = resnet_model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

```
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

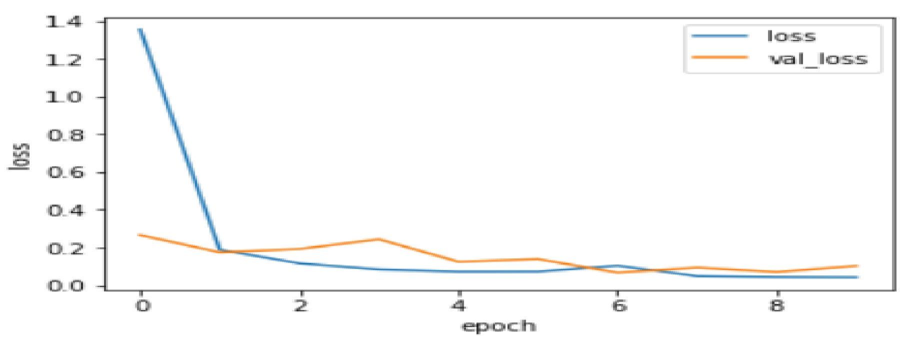
```
Accuracy on test data is:  97.36842105263158
```

- **Plot of Accuracy and Loss:**

```
# Plotting Loss & Accuracy Graphs
plt.figure(figsize=(12, 12))
plt.subplot(3, 2, 1)
plt.plot(history.history['accuracy'], label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.ylim(0,1)
plt.legend()
plt.show()
```




```
# Plotting Loss & Accuracy Graphs
plt.figure(figsize=(12, 12))
plt.subplot(3, 2, 2)
plt.plot(history.history['loss'], label = 'loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```



3.2 VGG16:

- **Model building:**

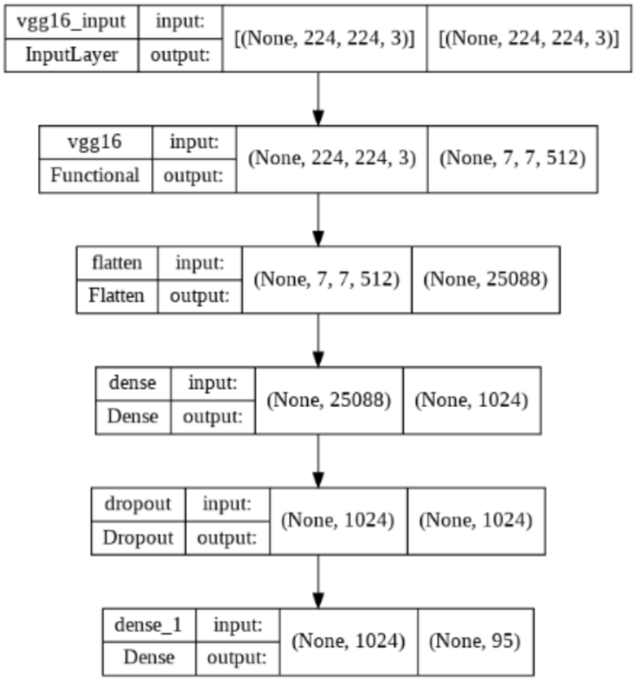
```
from tensorflow.keras.applications import vgg16
vgg_conv = vgg16.VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

```
# Freeze all the layers
for layer in vgg_conv.layers[:]:
    layer.trainable = False
 # Check the trainable status of the individual layers
for layer in vgg_conv.layers:
    print(layer, layer.trainable)
```

```
<keras.engine.input_layer.InputLayer object at 0x7f7a70489590> False
<keras.layers.convolutional.Conv2D object at 0x7f79f1726f90> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef7ce990> False
<keras.layers.pooling.MaxPooling2D object at 0x7f79ef6b5810> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6b6f90> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6bd7d0> False
<keras.layers.pooling.MaxPooling2D object at 0x7f79ef6c0350> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6c3c90> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6cb310> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6c3210> False
<keras.layers.pooling.MaxPooling2D object at 0x7f79ef656410> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef65d8d0> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef662f10> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6c3b90> False
<keras.layers.pooling.MaxPooling2D object at 0x7f79ef6b6f50> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef69c410> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef6b5b50> False
<keras.layers.convolutional.Conv2D object at 0x7f79ef66a510> False
```

```
# Create the the model
model = Sequential()
# Add the vgg16 convolutional base model
model.add(vgg_conv)

def VGG_16(weights_path=None):
    model = Sequential()
    model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
    model.add(Convolution2D(64, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(64, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))

    model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(128, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(128, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))

    model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(256, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(256, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(256, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))
```




```python
# Add new layers
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(95, activation='softmax'))
# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 1024)              25691136

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 95)                97375

=================================================================
Total params: 40,503,199
Trainable params: 25,788,511
Non-trainable params: 14,714,688
_________________________________________________________________
```

- **Compile Model:**

```python
model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['acc'])
```

```python
import keras
import tensorflow as tf
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

annealer = ReduceLROnPlateau(monitor='acc', factor=0.5, patience=3, verbose=1, min_lr=1e-4)
checkpoint = ModelCheckpoint('/temp/{epoch}_VGG16.h5', verbose=1, save_best_only=False, mode='auto', save_freq='epoch')
```

- ## Model training:

```
history = model.fit(
                  train_ds,
                  validation_data=validation_ds,
                  epochs=25,
                  batch_size=64,
                  callbacks=[annealer, checkpoint],
                  steps_per_epoch=len(train_ds),
                  validation_steps=len(test_ds)
                )
```

```
55/55 [==============================] - ETA: 0s - loss: 0.4831 - acc: 0.8926
Epoch 11: saving model to /temp/11_VGG16.h5
55/55 [==============================] - 22s 373ms/step - loss: 0.4831 - acc: 0.8926 - val_loss: 0.3726 - val_acc: 0.9375 - lr: 0.0010
Epoch 12/25
55/55 [==============================] - ETA: 0s - loss: 0.4407 - acc: 0.9060
Epoch 12: saving model to /temp/12_VGG16.h5
55/55 [==============================] - 21s 368ms/step - loss: 0.4407 - acc: 0.9060 - val_loss: 1.1180 - val_acc: 0.8906 - lr: 0.0010
Epoch 13/25
55/55 [==============================] - ETA: 0s - loss: 0.4610 - acc: 0.9035
Epoch 13: saving model to /temp/13_VGG16.h5
55/55 [==============================] - 22s 370ms/step - loss: 0.4610 - acc: 0.9035 - val_loss: 0.1395 - val_acc: 0.9688 - lr: 0.0010
Epoch 14/25
55/55 [==============================] - ETA: 0s - loss: 0.4452 - acc: 0.9015
Epoch 14: saving model to /temp/14_VGG16.h5
55/55 [==============================] - 22s 369ms/step - loss: 0.4452 - acc: 0.9015 - val_loss: 0.5715 - val_acc: 0.9219 - lr: 0.0010
```

- ## Evaluation:

```
model.evaluate(test_ds)
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.2835 - acc: 0.9211
[0.2834939658641815, 0.9210526347160339]
```

```
model.evaluate(train_ds)
```

```
55/55 [==============================] - 19s 323ms/step - loss: 0.0051 - acc: 0.9980
[0.005063485354185104, 0.9980005621910095]
```

- ## Testing:

```
y_pred=model.predict(test_ds)
y_pred.shape
```

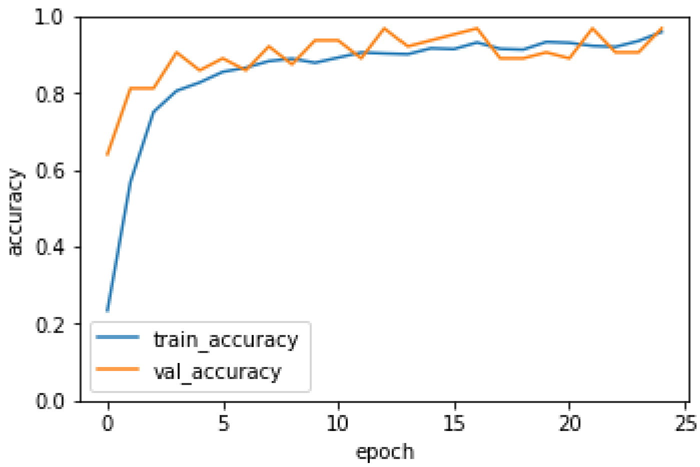
```
(38, 95)
```

```
class_names = train_ds.class_names
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

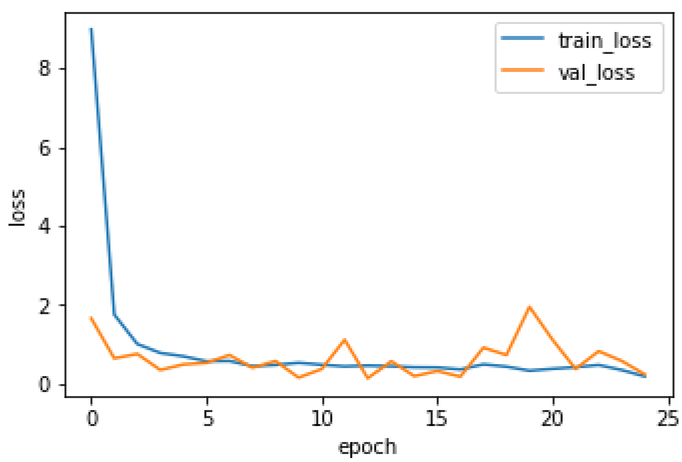
```
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

```
Accuracy on test data is:  92.10526315789474
```

- **Plot of Accuracy:**



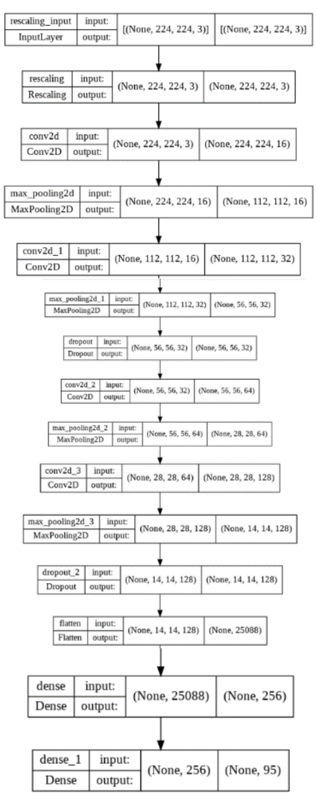
- **Plot of Loss:**



3.3 CNN:

- **Model building:**

```
cnn_model = Sequential([
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Conv2D(128, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(95, activation='softmax')
])
```



- **Model training:**

```
history = cnn_model.fit(train_ds,
                        batch_size=64,
                        validation_batch_size=64,
                        validation_data=validation_ds,
                        epochs=30)
```

```
Epoch 2/30
55/55 [==============================] - 11s 178ms/step - loss: 4.3004 - accuracy: 0.0191 - val_loss: 4.0759 - val_accuracy: 0.0457
Epoch 3/30
55/55 [==============================] - 11s 180ms/step - loss: 3.6842 - accuracy: 0.0891 - val_loss: 3.4204 - val_accuracy: 0.0914
Epoch 4/30
55/55 [==============================] - 11s 179ms/step - loss: 2.8490 - accuracy: 0.1879 - val_loss: 2.8132 - val_accuracy: 0.2171
Epoch 5/30
55/55 [==============================] - 11s 176ms/step - loss: 2.5364 - accuracy: 0.2622 - val_loss: 2.6485 - val_accuracy: 0.2543
Epoch 6/30
55/55 [==============================] - 11s 178ms/step - loss: 2.0588 - accuracy: 0.3602 - val_loss: 2.8886 - val_accuracy: 0.1714
Epoch 7/30
55/55 [==============================] - 12s 194ms/step - loss: 1.8346 - accuracy: 0.4267 - val_loss: 2.7336 - val_accuracy: 0.2657
Epoch 8/30
55/55 [==============================] - 11s 178ms/step - loss: 1.5546 - accuracy: 0.5159 - val_loss: 2.5471 - val_accuracy: 0.3114
Epoch 9/30
55/55 [==============================] - 11s 178ms/step - loss: 1.2907 - accuracy: 0.5898 - val_loss: 2.7216 - val_accuracy: 0.3571
Epoch 10/30
55/55 [==============================] - 11s 177ms/step - loss: 1.0238 - accuracy: 0.6750 - val_loss: 2.9759 - val_accuracy: 0.3171
Epoch 11/30
55/55 [==============================] - 11s 181ms/step - loss: 0.9574 - accuracy: 0.6929 - val_loss: 2.7835 - val_accuracy: 0.4200
Epoch 12/30
55/55 [==============================] - 11s 179ms/step - loss: 0.6498 - accuracy: 0.7875 - val_loss: 3.5382 - val_accuracy: 0.3171
```

- **Compile:**

```
cnn_model.compile(optimizer=Adam(lr=0.001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

- **Evaluation:**

```
cnn_model.evaluate(test_ds)
```

```
1/1 [==============================] - 0s 382ms/step - loss: 4.3269 - accuracy: 0.3158
[4.326930999755859, 0.31578946113586426]
```

```
cnn_model.evaluate(train_ds)
```

```
55/55 [==============================] - 9s 142ms/step - loss: 0.0411 - accuracy: 0.9900
[0.041087646037340164, 0.9900028705596924]
```

- ## Testing:

```
y_pred=cnn_model.predict(test_ds)
y_pred.shape
```

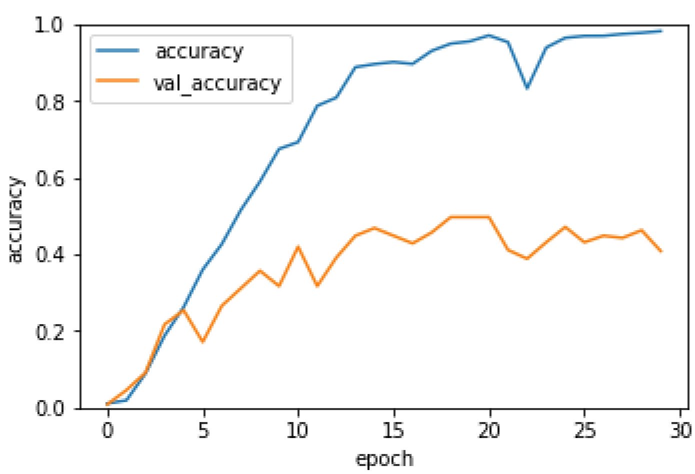
```
(38, 95)
```

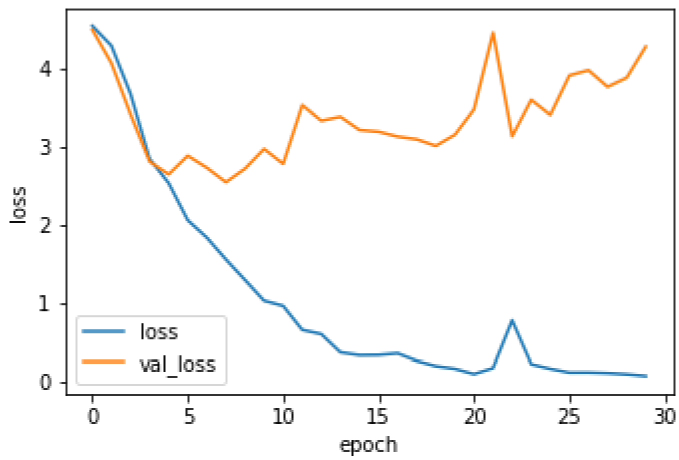
```
class_names = train_ds.class_names
actual = []
pred = []
for images, labels in test_ds:
    for i in range(0, len(images)):
        image = images[i]
        image = np.expand_dims(image, axis=0)
        result = cnn_model.predict(image)
        pred.append(class_names[np.argmax(result)])
        actual.append(class_names[labels[i].numpy()])
```

```
count=0
for i in range(len(actual)):
    if actual[i]==pred[i]:
        count+=1

print("Accuracy on test data is: ",(count/len(actual))*100)
```

```
Accuracy on test data is:  31.57894736842105
```

- ## Plot of Accuracy and Loss:





- **Models with dataset summary:**

Dataset	Image shape	Model type	epoch	Drop out	optimizer	loss	accuracy	validation loss	Validation accuracy	Test accuracy
Egyptian Hieroglyphic	(224,224,3)	VGG16	25	none	Adam	0.0051	0.9980	0.2835	0.9211	92.105
Egyptian Hieroglyphic	(224,224,3)	CNN	30	0.25	Adam	0.0411	0.9900	4.3269	0.3158	31.5789
Egyptian Hieroglyphic	(224,224,3)	Resnet50 (V1)	10	none	Adam	0.0958	0.9737	0.0599	0.9800	97.3684

- **Egyptian Hieroglyphic Conclusion:**

In this work, we have explored the capability of deep learning techniques to face the problem of ancient Egyptian hieroglyphs classification. Performances were measured using standard metrics, giving significant results for all the tested networks, in terms of performance as well as ease of training and computational saving.
In this view, the proposed work can be seen as the starting point for the implementation of much more complex goals. Actually, there are several open issues that may benefit from the use of the proposed approaches: coding, recognition and transliteration of hieroglyphic signs; recognition of determinatives and their semantic field; toposyntax of the hieroglyphic signs combined to form words; linguistics analysis of the hieroglyphic texts; recognition of corrupt, rewritten, and erased signs, towards even the identification of the “hand” of the scribe or the school of the sculptor.

References:

1. https://iq.opengenus.org/different-types-of-cnn-models/
2. https://aigents.co/data-science-blog/publication/introduction-to-convolutional-neural-networks-cnns
3. Z. Huang, X. Xu, J. Ni, H. Zhu, and C. Wang, “Multimodal representation learning for recommendation in Internet of Things,” *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10675–10685, 2019.View at: Publisher Site | Google Scholar
4. https://www.hindawi.com/journals/wcmc/2020/6677907/
5. C. Barat and C. Ducottet, “String representations and distances in deep convolutional neural networks for image classification,” Pattern Recognition, vol. 54, no. 6, pp. 104–115, 2016.View at: Publisher Site | Google Scholar
6. H. Liang, A. Xian, M. Min Mao, P. Ni, and H. Wu, “A research on remote fracturing monitoring and decision-making method supporting smart city,” *Sustainable Cities and Society*, vol. 62, article 102414, 2020.View at: Publisher Site | Google Scholar
7. https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/
8. https://link.springer.com/article/10.1007/s12652-021-03488-z
9. https://huggingface.co/Axon/resnet50-v1
10. ResNetv1 Deep residual learning for image recognition He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.
11. https://cv-tricks.com/keras/understand-implement-resnets/
12. https://ieeexplore.ieee.org/document/9528382/references#references