SAPIENZA
UNIVERSITÀ DI ROMA

Department of Computer, Automatic and Management Engineering

Machine Learning HW02
Image Classification

Pouya Irani
Matricola: 1937388
Email: irani.1937388@studenti.uniroma1.it

Table of Contents

1 Dataset description 3

2 Classification Problem ..... Error! Bookmark not defined.

2.1 Preprocessing 3

2.2 Over sampling 4

2.2.1 SMOTE ..... Error! Bookmark not defined.

2.3 Grid search ..... Error! Bookmark not defined.

2.4 KNN ..... Error! Bookmark not defined.

2.5 SVM ..... Error! Bookmark not defined.

3 Regression Problem ..... Error! Bookmark not defined.

3.1 Preprocessing (SMOGN) ..... Error! Bookmark not defined.

3.2 Regression algorithms ..... Error! Bookmark not defined.

3.2.1 Linear Regression ..... Error! Bookmark not defined.

3.2.2 Random forest ..... Error! Bookmark not defined.

3.2.3 Polynominal Regression ..... Error! Bookmark not defined.

3.2.4 Voting Regressor ..... Error! Bookmark not defined.

3.2.5 Results 9

1 Dataset description

The dataset is included images of 10 different animals which each class has more than 300 sample images.

2 Aim of the project

The goal of the project is to obtain a model to classify each of the animal classes based on the dataset.

2.1 Preprocessing

Before performing any pre-processing model, we need to analyze the dataset to realize the number of classes and data distribution.
Thou the classes are as follows:

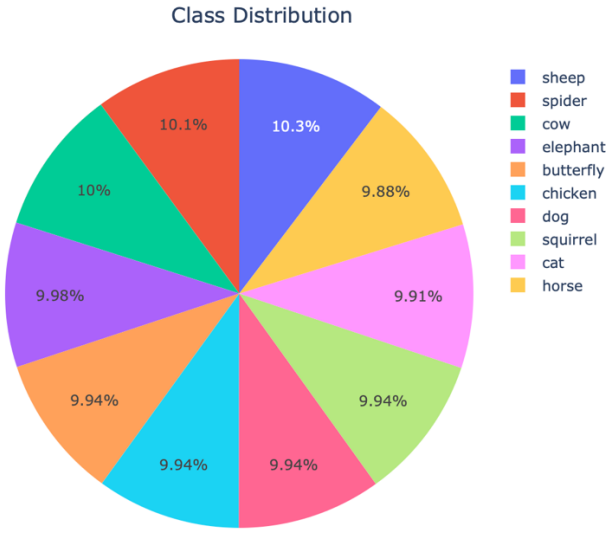

figure 1 data distribution

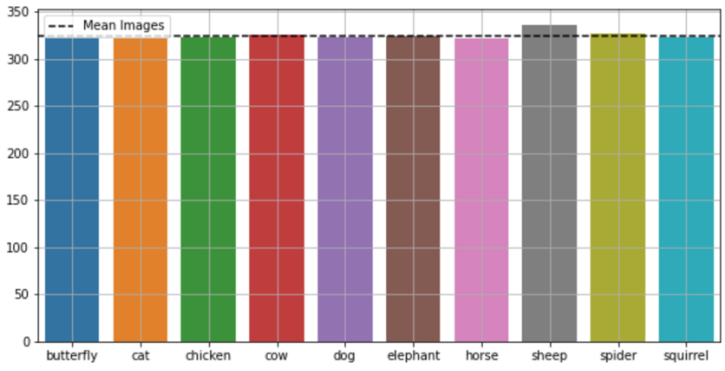

figure 2 number of samples in each class

As it clearly visible the classes are balanced and now it is possible to continue to do the pre-processing procedure and prepare the data to fit the models.

2.2 Over sampling

Although the data is balance there is a high probability of low accuracy since the number of samples for each class is too low and there is a massive variety in types of samples which make the prediction difficult for the model.

2.2.1 Image Generator

One approach to addressing the mentioned problem is to use a standard tool in tensorflow keras in python.

Keras ImageDataGenerator is used for getting the input of the original data and further, it makes the transformation of this data on a random basis and gives the output resultant containing only the data that is newly transformed. It does not add the data. Keras image data generator class is also used to carry out data augmentation where we aim to gain the overall increment in the generalization of the model. Operations such as rotations, translations, shearin, scale changes, and horizontal flips are carried out randomly in data augmentation using an image data generator.

Keras image data generator is used for the generation of the batches containing the data of tensor images and is used in the domain of real-time data augmentation. We can loop over the data in batches when we make use of the image data generator in Keras. There are various methods and arguments of the image data generator class that helps to define the behavior of the data generation.

Figure 3 demonstrates the application of the image generator.

```
# Initialize Generator
gen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    vertical_flip=True,
    rotation_range=20,
    validation_split=0.2)
```

figure 3image generator application

Now we can split the data into training and validation sets.

```
# Load data
train_ds = gen.flow_from_directory(
    path,
    target_size=(256,256),
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    subset='training')

valid_ds = gen.flow_from_directory(
    path,
    target_size=(256,256),
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    subset='validation')
```

```
Found 2603 images belonging to 10 classes.
Found 645 images belonging to 10 classes.
```

figure 4 train - validation split

2.3 Sequential Model

Tensorflow sequential is the group containing the stack of linear format that consists of various layers of the library package tf.keras.Model. This Sequential class is inherited from the Module, Layer, and Model classes. The basic functionality of Sequential is to make the provision of inferences and training of the module.

Tensorflow Sequential model can be implemented by using Sequential API. The methodology followed while building the model is step by step and working on a single layer at a particular time. The Sequential tensorflow API is the easiest way using which we can run and create the keras models. Along with that, it also brings in certain limitations while creating the models.

We call the model sequential because creating the model involves creating and defining the class of sequential type and specifying the layers to be added to the model. The layer addition is done step by step, which means one layer simultaneously, from input to output.

2.3.1 Softmax Activation function

Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

Figure 4 demonstrates the designed model which contains all the specification based on dataset inputs and desired output such as image input size(256,256) in 3 RGB colors, and also last layer output neurons is equal to number of classes.

```
num_classes = 10

model = tf.keras.Sequential([
  #tf.keras.layers.Rescaling(1./127),
  tf.keras.layers.Conv2D(64, 3, activation='relu', input_shape=(256,256,3)),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(128, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(512, activation='relu'),


  tf.keras.layers.Dense(num_classes,activation="softmax")
])
```

figure 5 designed model

2.3.2 Cross-Entropy Loss Function

Working with Machine learning and Deep learning models involve usage of cost functions which are there to optimize the model during the training. Better is the model, the lower will be the loss. One of the most used cost function for classification based problem statement is Cross-Entropy.
Cross-Entropy Loss is also known as logarithmic loss, log loss or logistic loss. Each probability of the predicted class is compared with the actual class and loss is calculated which penalizes the probability based on how far it is from the actual expected value. The penalty is logarithmic in nature yielding a large score for large differences close to 1 and small score for small differences tending to 0. A perfect model has a cross-entropy loss of 0.

Cross-entropy is defined as

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

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

Both categorical cross entropy and sparse categorical cross-entropy have the same loss function as defined above. The only difference between the two is on how labels are defined.

- Categorical cross-entropy is used when we have to deal with the labels that are one-hot encoded, for example, we have the following values for 3-class classification problem **[1,0,0], [0,1,0]** and **[0,0,1]**.

- In sparse categorical cross-entropy , labels are integer encoded, for example, **[1], [2]** and **[3]** for 3-class problem.

In this case we use sparse categorical loss function.

```
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

figure 6 model compile

2.3.3 Results

Figure 7 and 8 testify the model after fitting the model to the inputs with 20 epochs.
It is clearly visible that after epoch number 16 although the training accuracy increases there is no significant change in validation accuracy and the same fact is true about the loss function.

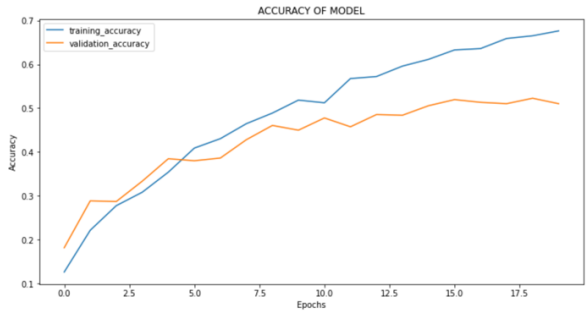

figure 7accuracy of the model

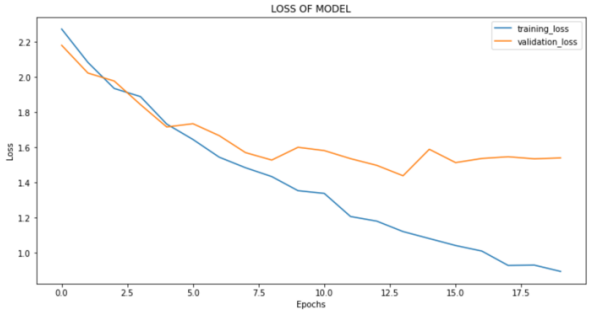

figure 8 loss of the model

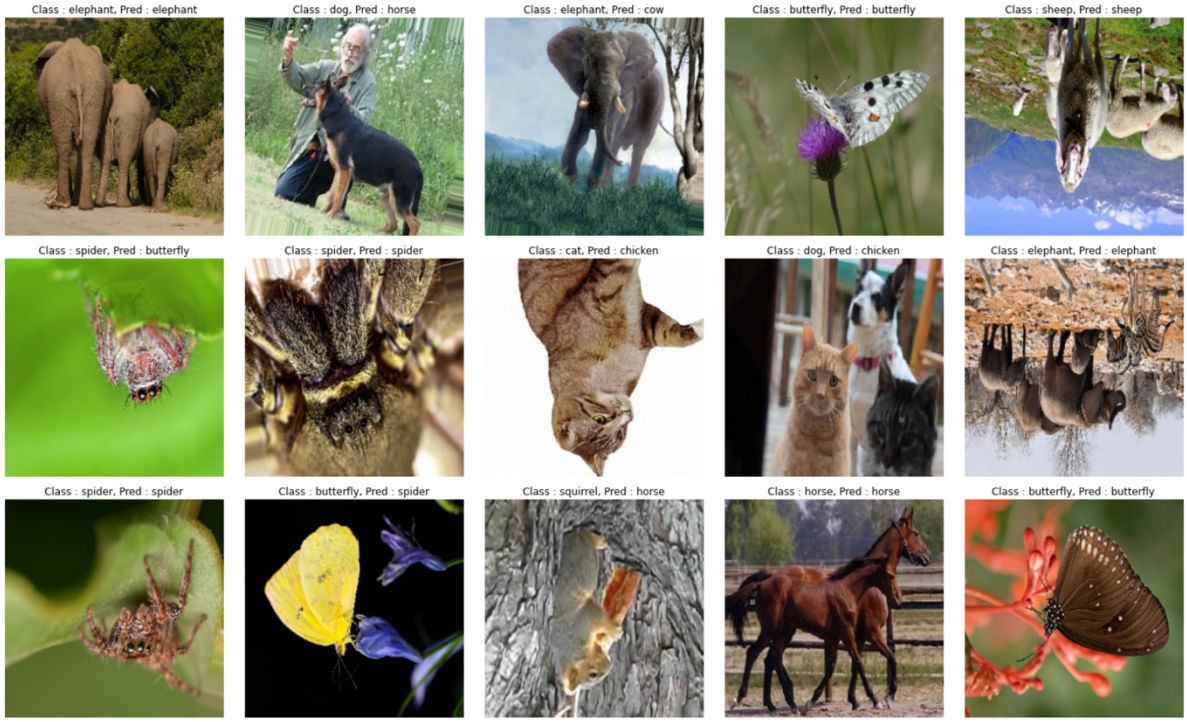

figure 9 testing the model

3 Pre-trained model

To compare designed model with more advanced models in this part I used Xception to have a better perspective of the problem.

3.1 Xception

Xception is a convolutional neural network. Xception is the pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals etc…

3.1.1 Xception architecture

Architecture Xception, which stands for “Extreme Inception”. The Xception architecture has 36 convolutional layers forming the feature extraction base of the network.

The data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and Separable Convolution layers are followed by batch normalization (not included in the diagram). All Separable Convolution layers use a depth multiplier of 1 (no depth expansion).

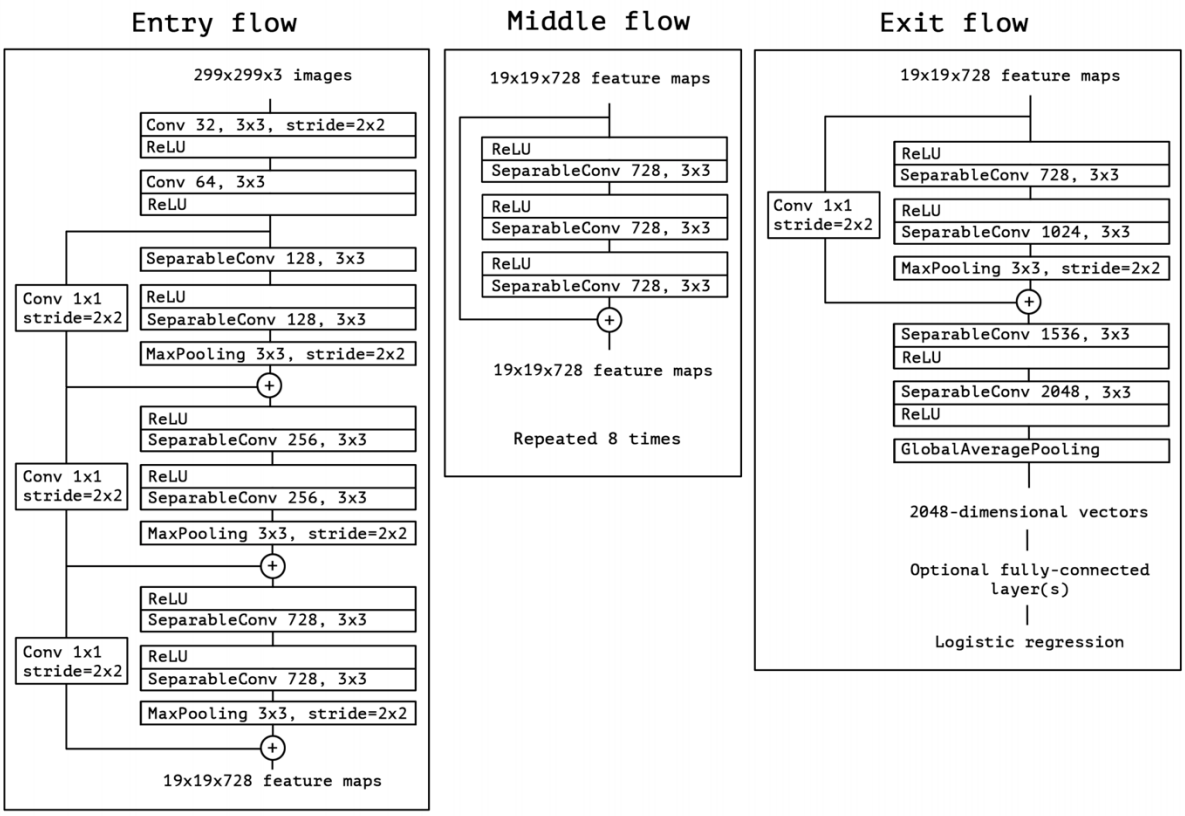

figure 10 Xception architecture

3.1.2 Xception implementation

Figure 11 shows the imolementation of Xception by specifying the parameter close to the designed sequential model.

```
# Specify Model Name
name = "Xception"

 # Pretrained Model
base_model = Xception(include_top=False, input_shape=(256,256,3), weights='imagenet')
base_model.trainable = False # Freeze the Weights

 # Model
xception = Sequential([
     base_model,
     GAP(),
     Dense(256, activation='relu'),
     Dropout(0.2),
     Dense(n_classes, activation='softmax')
 ], name=name)

 # Compile
xception.compile(
     loss='sparse_categorical_crossentropy',
     optimizer='adam',
     metrics=['accuracy']
 )

 # Callbacks
cbs = [
     EarlyStopping(patience=3, restore_best_weights=True),
     ModelCheckpoint(name + ".h5", save_best_only=True)
 ]

 # Train Model
hist = xception.fit(
     train_ds, validation_data=valid_ds,
     epochs=10, callbacks=cbs
 )
```

figure 11Xception implementation

3.1.3 Results

Below has shown the outcome of the Xception

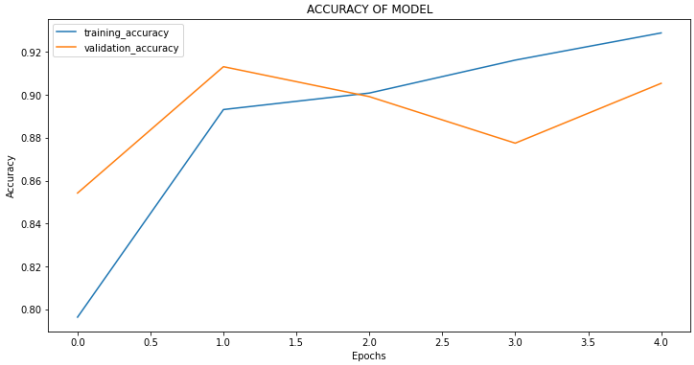

figure 12 accuracy

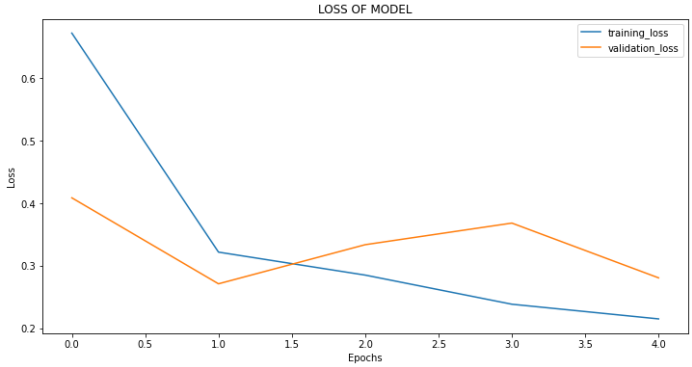

figure 13 loss

figure 14teste images

Since the architecture of the Xception is more advanced the results are much more accurate both in terms of accuracy and loss function and also the tested images above. On the other hand it reached higher accuracy in less epochs which means it requires less time and computational effort.