Animal Classification - CNN

DATS 6203

Team 04

Shuting Cai, Jinbo Li, Kaiyuan Liang

Professor: Amir Jafari

Abstract

Convolutional Neural Network is one of the common ways applied to analyze visual imagery. The team was planning to build a CNN using TensorFlow, which is suitable for animal classification by finding the best model with the goal of the highest accuracy in testing. Based on the list of metrics results, accuracy score, micro f1 score, and cohen kappa score, the team found that Resnet50 is the best fit in our case, which runs 93% accuracy, 0.92 cohen kappa score, and 0.93 micro f1 scores. The team recommended using the Resnet50 pretrained model in this dataset, trying various hyperparameters, and building the model in a machine with larger memories.

Introduction

Image recognition is a popular technology, which can accurately identify the visual content in the image, and provide a variety of objects, scenes, and concept labels, with the ability of target detection and attribute recognition, to help clients accurately identify and understand the image. The technique has been applied to multiple fields such as text recognition, license plate recognition, and face recognition. The team would explore the topic of computer vision primarily, applying the image classification technique to animal detection. The animal classification could be used in animal management and tag the different animal species, for further application, it could be developed and suitable for animal photo recognition, early childhood education science, and image content analysis, to help people get a better idea of animals and the diversity of nature. The team's objectives are to design a CNN model with 90% accuracy, which will be accomplished by exploring the architecture of the CNN model and comparing the state-of-the-art pre-trained models and the model with the customized architecture. The team is using TensorFlow to implement the network, for the reason that TensorFlow is a powerful and mature deep learning library with strong visualization capabilities, and there are multiple options for advanced model development. With the data augmentation of the training, the data will expand up the training dataset, therefore, the model can be fitted better. The hyperparameters that learning rate, number of epochs, batch size also are determined experimentally to achieve the highest accuracy.

Description of the data set

The team is going to use the "Animal Image Classification Dataset," a source from Kaggle, which contains 12 classes of animals and at least 1,200 image files for each class (https://www.kaggle.com/datasets/piyushkumar18/animal-image-classification-dataset), and the images in the dataset have large variations in scale, pose, background and lighting, which is enough to train the common architectures in the convolutional neural network, besides, data augmentation would be used, it will expand up the training dataset.

Before we started, we used a package named split-folders to split the dataset into the training dataset, validation dataset, and testing dataset. And we did exploratory data analysis to check if the dataset is balanced or not.

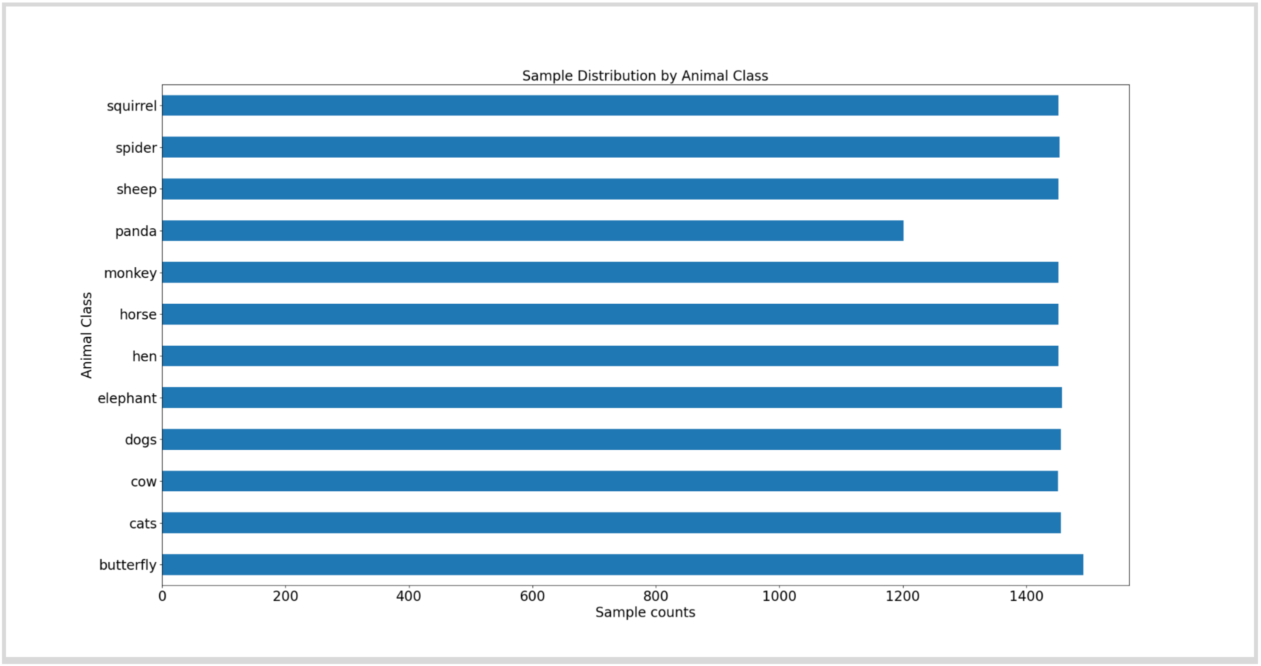


Figure1. Dataset distribution by classes.

As we can see from the EDA plots, the datasets are almost balanced.

Description of the deep learning network and training algorithm

There are three key components in the typical architecture of the Convolutional Neural Network. First of all, Convolutional Layer is the core building block of a CNN, the layer's parameters consist of a set of learnable filters, which have a small receptive field, but extend through the full depth of the input volume. Compared with the convolutional layer, the pooling layer can reduce the data dimension more effectively, which can not only greatly reduce the amount of computation, but also effectively avoid overfitting. The pooling layer is followed by a convolutional layer, it can subsample a convolutional layer to reduce the number of parameters in a CNN and the time and space complexity. The fully connected layer is similar to the part of the traditional neural network and is used to output the desired result.

ResNet was the winner of the 2015 ImageNet ILSVRC challenge with a top-five error rate under 3.6%. ResNet50 can work with 50 deep neural network layers. The key building blocks in ResNet are Deep Residual Networks. Deep residual nets make use of residual blocks to improve the accuracy of the models.

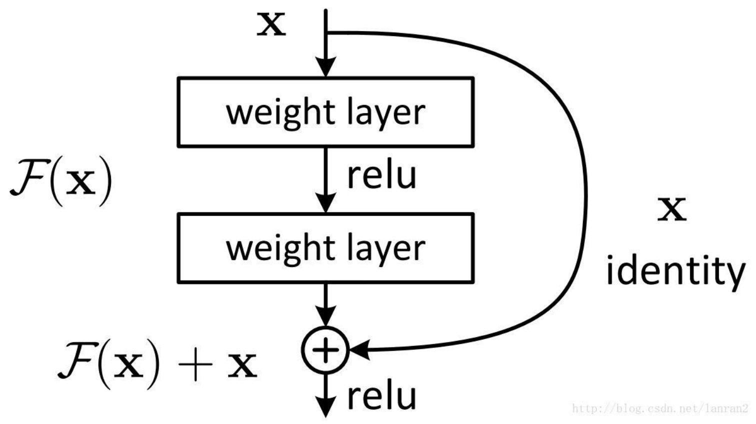

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
http://blog.csdn.net/lanran2

Figure 2. The skip connection.

In each residual unit, the number of sublayers in the convolutional layer is doubled, and the height and width of the sublayers are halved with 2 strides. The concept of “skip connections,” which lies at the core of the residual blocks, is the strength of this type of neural network. From figure 2, the curve is “skip connection”, and the other is residual mapping, y = F(x) +x, then F(x) = y-x, that is the residual. When the model is the most optimal, the residual mapping will be pushed to 0 then only identity mapping works, to keep the networks optimal. Compare with Resnet34, a 3-layer bottle-neck block instead of 2-layer blocks, Resnet 50 architecture can save the running time and has much higher accuracy than the 34-layer ResNet model. The architecture of ResNet is using ReLU as the activation function and is followed by a batch normalization layer. The team freezing a layer of the pre-trained architecture, freezing layer is a technique to accelerate neural network training by progressively freezing hidden layers, which can help the model to get better performance.

VGG19

VGGNet is a Convolutional Neural Network architecture proposed by Karen Simonyan and Andrew Zisserman from the University of Oxford in 2014 (Nepal, 2020). And we used the VGG19 model. The creators of this model evaluated the networks and enhanced the depth using an architecture with very small (3 × 3) convolution filters, which has a significant improvement on the configurations.

Architecture of VGG19

There are 16 convolutional layers, 3 fully connected layers, 5 max-pooling layers, and 1 softmax layer in VGG19. Although summing up to 24 layers in VGG19, it has only 19 weight layers (learnable parameters layers). The input size of VGG19 is 224 by 224 with 3 RGB channels. VGG19 used kernels of 3 by 3 size with a stride size of 1 pixel and max-pooling was performed over 2 by 2-pixel windows with stride 2. And it implemented three fully connected layers from which the first two were of size 4096 and after that, a layer with 1000 channels for 1000-way ILSVRC classification, and the final layer is a softmax function. And the depth is the number of filters.

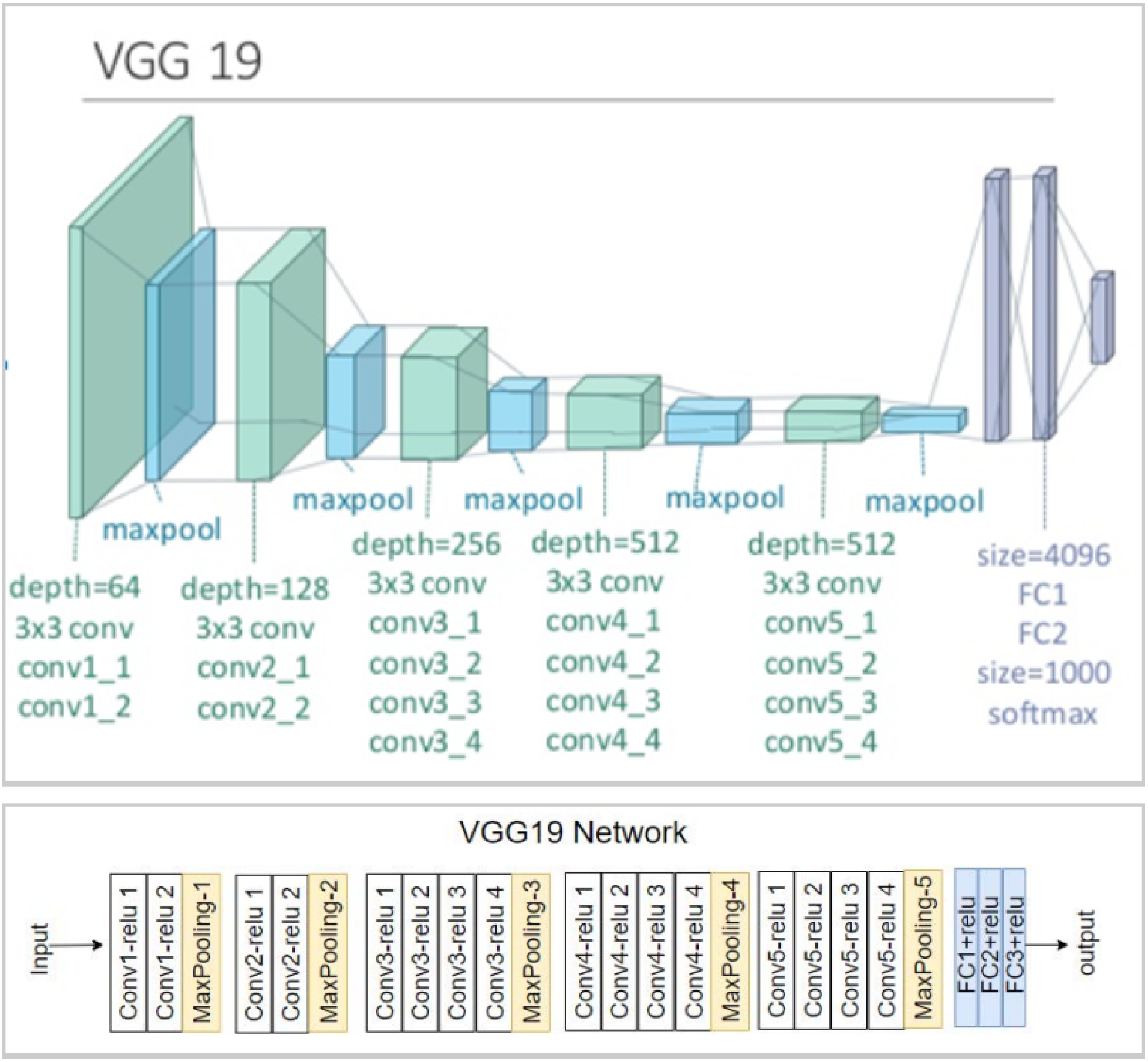


Figure 3. VGG19 model architecture and networks.

Experimental setup

The original dataset would be split into training, validation, and testing dataset. In training, the loss and accuracy of train and validation would be detected; In testing, the predicted result would be used in judging the performance of the network. The team split the train and test dataset into subfolders with a ratio of 9 to 1, and then used *tf.keras.utils.image_dataset_from_directory* to split the training and validation dataset with a ratio of 9 to 1. Therefore, we were having 10848 image files in training, 4649 image files in the validation dataset, and 1731 image files in testing.

In the part of data augmentation, randomly flipping the image, randomly rotating the image, and resizing and rescaling the images are good ways to do the data augmentation in training, which can help the network to learn from training without overfitting. There is a disadvantage of data augmentation that is increasing data bias (Xu, 2020). If the augmented data distribution would be significantly different from the original dataset, the performance of the network would be getting worse.

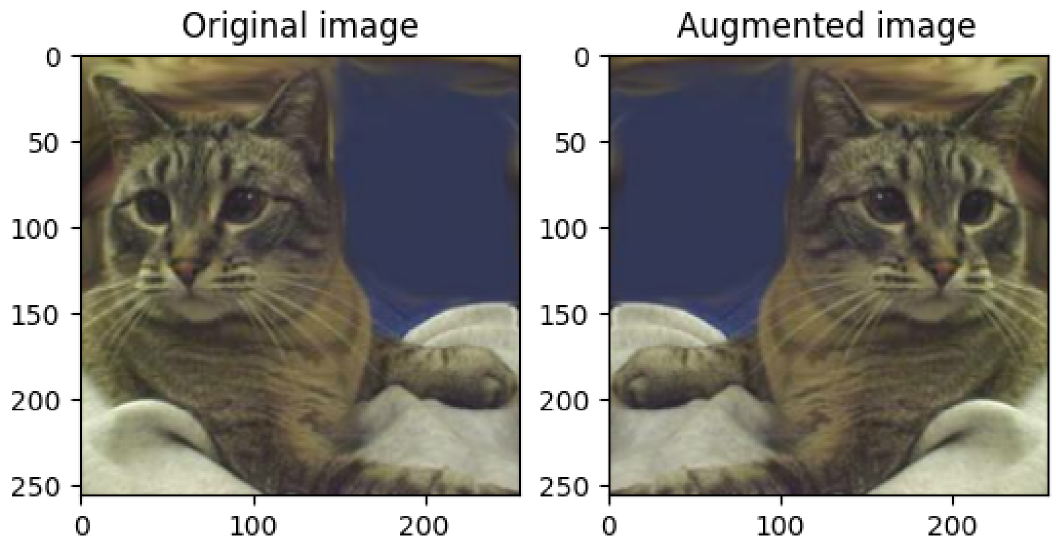


Figure 4. The original image vs. the horizontally flipped images.

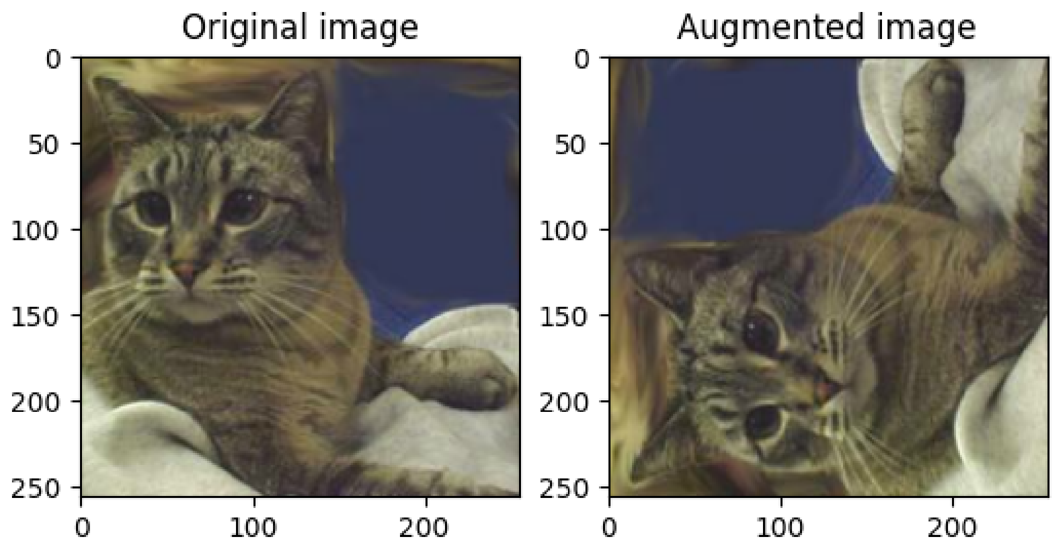


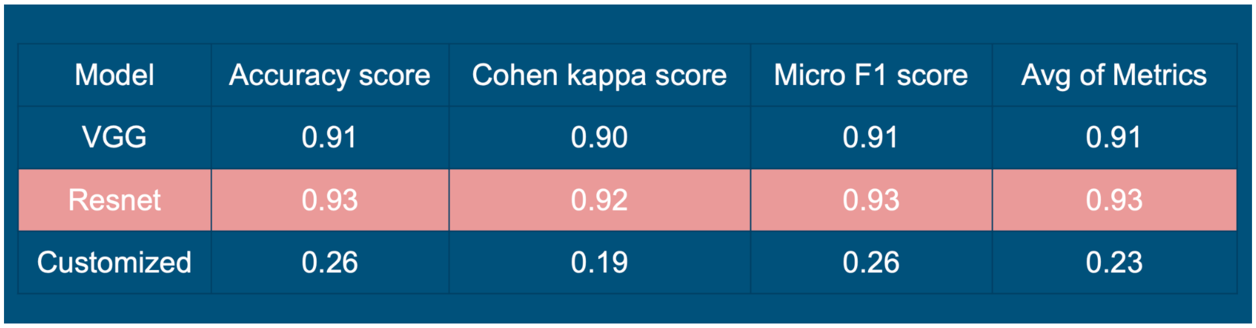
Figure 5. The original image vs. the 90 degrees rotated images.

In the list of evaluation metrics, the team would use the accuracy score, cohen kappa score, micro F1 score, and hamming loss. The accuracy score is one of the most straightforward metrics in machine learning, which can tell how accurate the model is, and how the predicted result matches the target. Cohen kappa score is one of the evaluation indicators, which can be used for binary or multi-class problems, but it is not suitable for multi-label. It indicates a good classification when the cohen kappa score is higher than 0.8. The F1 score is a popular metric for evaluating the performance of classification. A higher F1 score shows a better classification. The macro F1 score will compute the metric independently for each class and then take the average, while the micro F1 score aggregates the contributions of all classes to compute the average metric (SHASHANK). In the present case, the micro-averaging F1 score is preferable to be used as an evaluation indicator in this multiple-class problem. The Hamming distance is also suitable for multi-classification problems, it measures the distance between the predicted label and the true label, and the value is between 0 and 1. A distance of 0 means that the predicted result is exactly the same as the real result, whereas, a distance of 1 means that the predicted result is completely contrary to the true target.

The learning rate defines how quickly a network updates its parameters. A low learning rate slows down the learning process but converges smoothly. A larger learning rate speeds up the learning but may not converge. A decaying Learning rate is preferred. The number of epochs is the number of times the whole training data is shown to the network while training. Increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting). Mini batch size is the number of subsamples given to the network after which parameter update happens. Softmax is the transformed function because of the multiple class problem.

Consider the situation of underfitting and overfitting, underfitting will happen when the dataset is not big enough or the model is not powerful enough, and the network has not learned the relevant patterns in the training data (*Overfit and underfit*). Along with the training process, overfitting might occur, the network has learned the patterns from training data that fails to generalize the testing dataset. The way to avoid overfitting is either by using the appropriate number of epochs or by applying the regularization techniques. To detect overfitting or underfitting, the team compares the validation metrics to the training metrics that show the trend of accuracy and loss for training and validation. If both metrics are moving in the same direction, everything is fine; If the validation metric begins to stagnate while the training metric continues to improve, you are probably close to overfitting; If the validation metric is going in the wrong direction, the model is clearly overfitting (*Overfit and underfit*).

Results and Improvement



Model	Accuracy score	Cohen kappa score	Micro F1 score	Avg of Metrics
VGG	0.91	0.90	0.91	0.91
Resnet	0.93	0.92	0.93	0.93
Customized	0.26	0.19	0.26	0.23

Table1. Model Evaluation.

From the result table, we can see that VGG19 and Resnet perform reasonably well and Resnet is slightly better than VGG19. However, the Customized model's metrics are rather low, which still have some space to improve in the future. Above all, the Resnet50 is the best model in our dataset.

There are still some areas that we can improve in the future. Firstly, we can build the model with more memories so that it will not get out of memory error. Secondly, we should test various hyperparameter settings to optimize the customized model to get a better result. Lastly, We could try different pre-trained models such as GoogleNet to compare with different models to get optimal results.

References

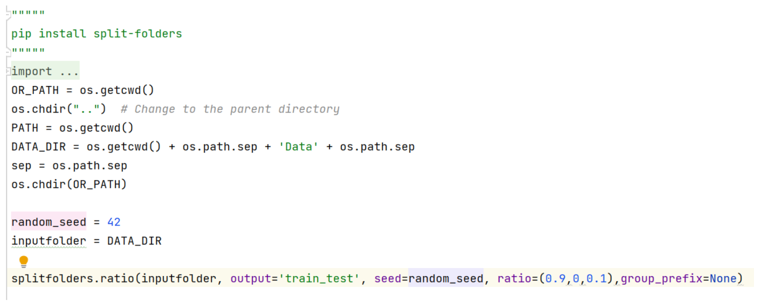
SHASHANK GUPTASHASHANK GUPTA 3, pythiestpythiest 4, Rahul Reddy VemireddyRahul Reddy Vemireddy 74544 silver badges66 bronze badges, David MakovozDavid Makovoz 42044 silver badges99 bronze badges, Saghan MudbhariSaghan Mudbhari 5111 silver badge11 bronze badge, Sujit JenaSujit Jena 3122 bronze badges, Lerner ZhangLerner Zhang 34611 silver badge88 bronze badges, goyuiitvgoyuiitv 1133 bronze badges, & Olivier D'AnconaOlivier D'Ancona 10122 bronze badges. (1964, October 1). *Micro average vs Macro average performance in a multiclass classification setting*. Data Science Stack Exchange. Retrieved April 22, 2022, from https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin

Nepal, P. (2020, December 8). *VGGNet architecture explained*. Medium. Retrieved April 22, 2022, from https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6#:~:text=VGGNet%20is%20a%20Convolutional%20Neural%20Network%20architecture%20proposed%20by%20Karen,network%20depth%20on%20its%20accuracy.

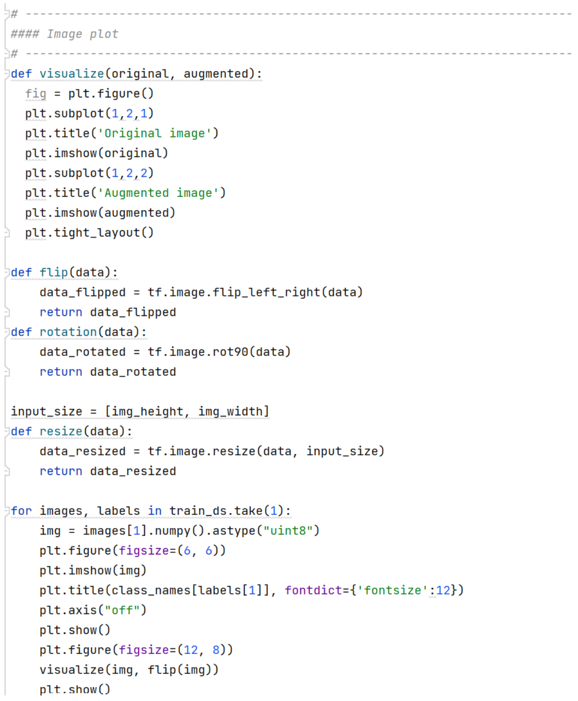
Overfit and underfit: Tensorflow Core. TensorFlow. (n.d.). Retrieved April 22, 2022, from https://www.tensorflow.org/tutorials/keras/overfit_and_underfit

Xu, Y., Noy, A., Lin, M., Qian, Q., Li, H., & Jin, R. (2020, October 3). *WeMix: How to better utilize data augmentation*. arXiv.org. Retrieved April 25, 2022, from https://arxiv.org/abs/2010.01267#:~:text=The%20main%20limitation%20of%20data,of%20existing%20data%20augmentation%20methods.

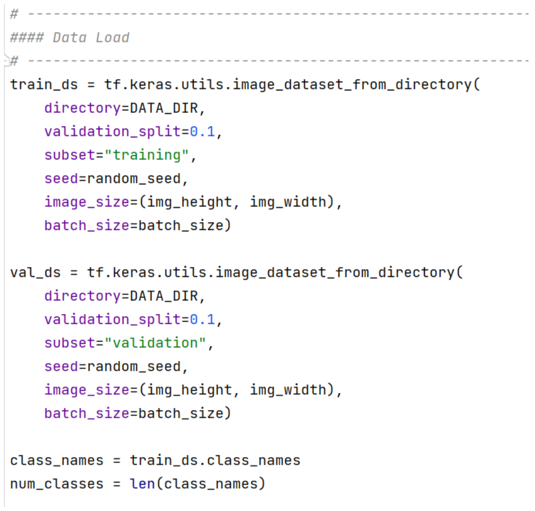
Appendix



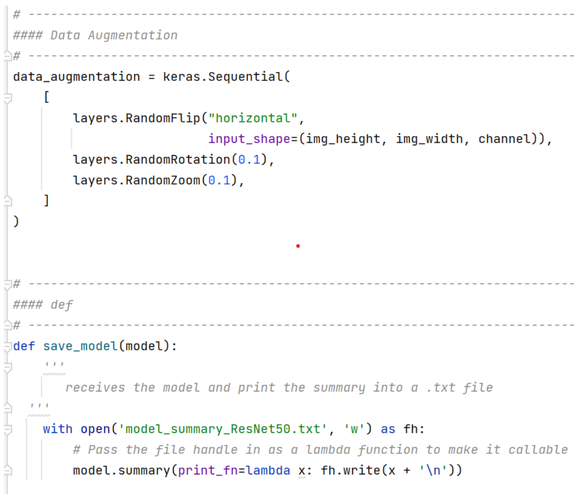
```
"""
pip install split-folders
"""
import ...
OR_PATH = os.getcwd()
os.chdir("..")  # Change to the parent directory
PATH = os.getcwd()
DATA_DIR = os.getcwd() + os.path.sep + 'Data' + os.path.sep
sep = os.path.sep
os.chdir(OR_PATH)

random_seed = 42
inputfolder = DATA_DIR

splitfolders.ratio(inputfolder, output='train_test', seed=random_seed, ratio=(0.9,0,0.1),group_prefix=None)
```



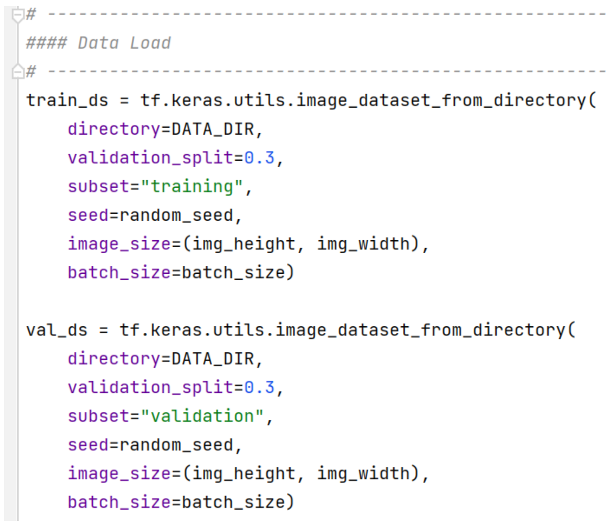
```
# ------------------------------------------------------------------------
#### Image plot
# ------------------------------------------------------------------------
def visualize(original, augmented):
  fig = plt.figure()
  plt.subplot(1,2,1)
  plt.title('Original image')
  plt.imshow(original)
  plt.subplot(1,2,2)
  plt.title('Augmented image')
  plt.imshow(augmented)
  plt.tight_layout()

def flip(data):
    data_flipped = tf.image.flip_left_right(data)
    return data_flipped
def rotation(data):
    data_rotated = tf.image.rot90(data)
    return data_rotated

input_size = [img_height, img_width]
def resize(data):
    data_resized = tf.image.resize(data, input_size)
    return data_resized

for images, labels in train_ds.take(1):
    img = images[1].numpy().astype("uint8")
    plt.figure(figsize=(6, 6))
    plt.imshow(img)
    plt.title(class_names[labels[1]], fontdict={'fontsize':12})
    plt.axis("off")
    plt.show()
    plt.figure(figsize=(12, 8))
    visualize(img, flip(img))
    plt.show()
```



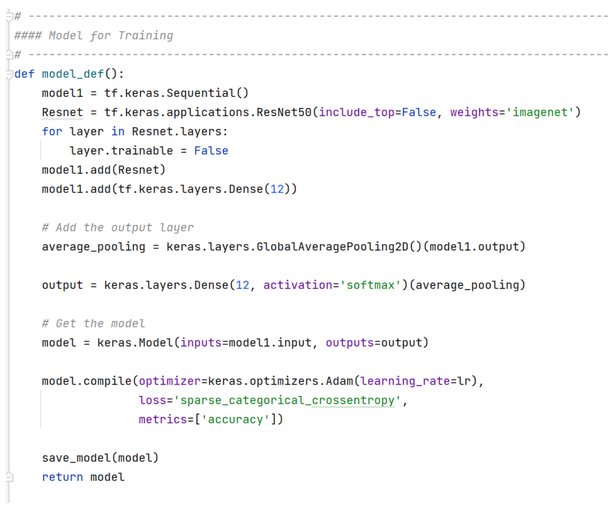
```
# ------------------------------------------------------------
#### Data Load
# ------------------------------------------------------------
train_ds = tf.keras.utils.image_dataset_from_directory(
    directory=DATA_DIR,
    validation_split=0.1,
    subset="training",
    seed=random_seed,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    directory=DATA_DIR,
    validation_split=0.1,
    subset="validation",
    seed=random_seed,
    image_size=(img_height, img_width),
    batch_size=batch_size)

class_names = train_ds.class_names
num_classes = len(class_names)
```



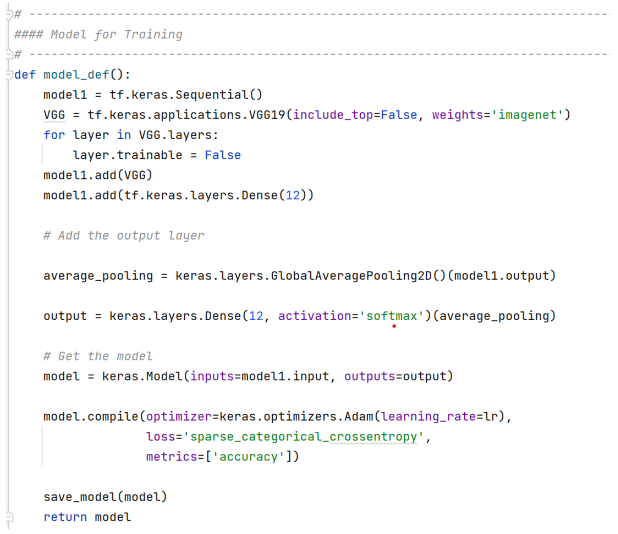
```
# ------------------------------------------------------------------------
#### Data Augmentation
# ------------------------------------------------------------------------
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal",
                          input_shape=(img_height, img_width, channel)),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
    ]
)


# ------------------------------------------------------------------------
#### def
# ------------------------------------------------------------------------
def save_model(model):
    '''
       receives the model and print the summary into a .txt file
  '''
    with open('model_summary_ResNet50.txt', 'w') as fh:
        # Pass the file handle in as a lambda function to make it callable
        model.summary(print_fn=lambda x: fh.write(x + '\n'))
```



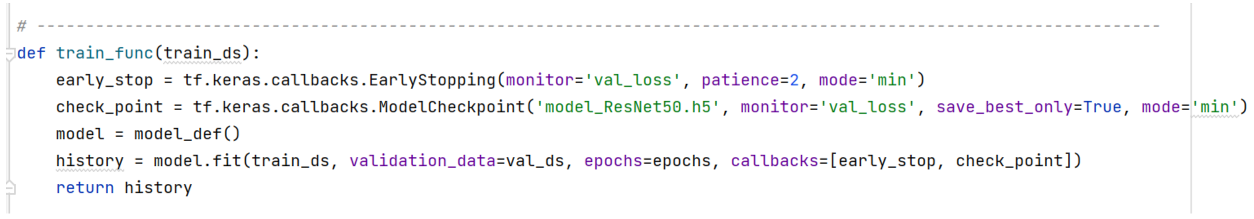
```
# --------------------------------------------------------
#### Data Load
# --------------------------------------------------------
train_ds = tf.keras.utils.image_dataset_from_directory(
    directory=DATA_DIR,
    validation_split=0.3,
    subset="training",
    seed=random_seed,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    directory=DATA_DIR,
    validation_split=0.3,
    subset="validation",
    seed=random_seed,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```



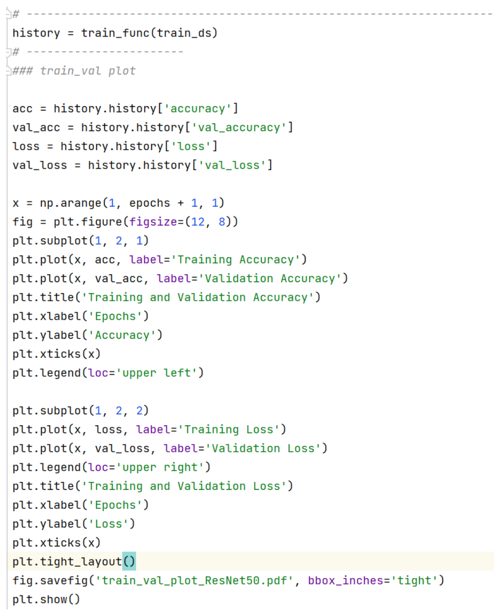
```
# ------------------------------------------------------------------------------------
#### Model for Training
# ------------------------------------------------------------------------------------
def model_def():
    model1 = tf.keras.Sequential()
    Resnet = tf.keras.applications.ResNet50(include_top=False, weights='imagenet')
    for layer in Resnet.layers:
        layer.trainable = False
    model1.add(Resnet)
    model1.add(tf.keras.layers.Dense(12))

    # Add the output layer
    average_pooling = keras.layers.GlobalAveragePooling2D()(model1.output)

    output = keras.layers.Dense(12, activation='softmax')(average_pooling)

    # Get the model
    model = keras.Model(inputs=model1.input, outputs=output)

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=lr),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    save_model(model)
    return model
```



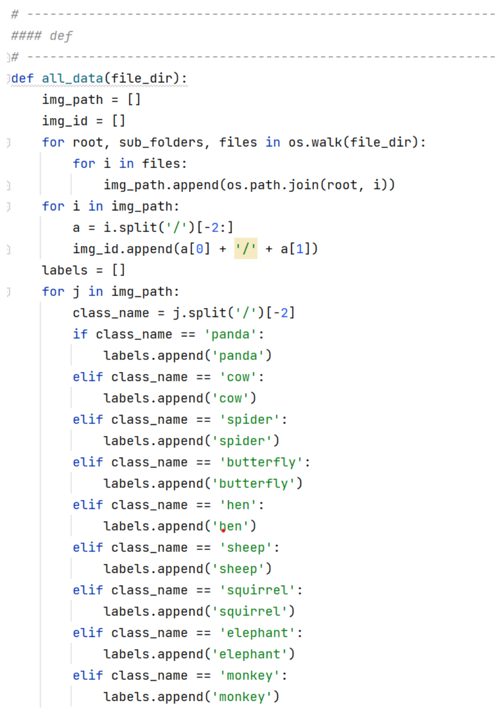
```
# ------------------------------------------------------------------------------------
#### Model for Training
# ------------------------------------------------------------------------------------
def model_def():
    model1 = tf.keras.Sequential()
    VGG = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
    for layer in VGG.layers:
        layer.trainable = False
    model1.add(VGG)
    model1.add(tf.keras.layers.Dense(12))

    # Add the output layer

    average_pooling = keras.layers.GlobalAveragePooling2D()(model1.output)

    output = keras.layers.Dense(12, activation='softmax')(average_pooling)

    # Get the model
    model = keras.Model(inputs=model1.input, outputs=output)

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=lr),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    save_model(model)
    return model
```



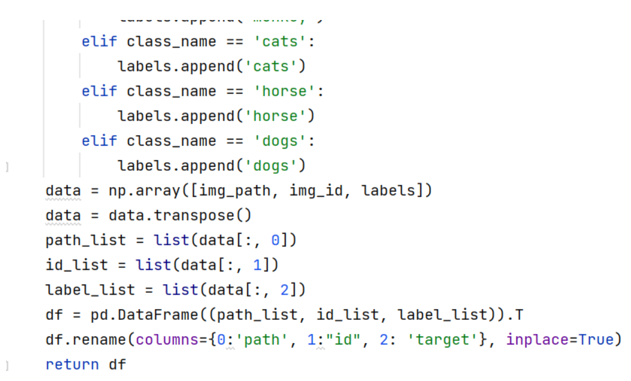
```
# ------------------------------------------------------------------------------------------------------------
def train_func(train_ds):
    early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2, mode='min')
    check_point = tf.keras.callbacks.ModelCheckpoint('model_ResNet50.h5', monitor='val_loss', save_best_only=True, mode='min')
    model = model_def()
    history = model.fit(train_ds, validation_data=val_ds, epochs=epochs, callbacks=[early_stop, check_point])
    return history
```



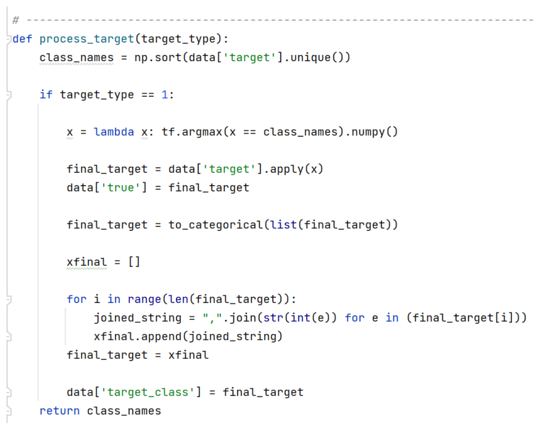
```
# ------------------------------------------------------------------------
history = train_func(train_ds)
# ----------------------
### train_val plot

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

x = np.arange(1, epochs + 1, 1)
fig = plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.plot(x, acc, label='Training Accuracy')
plt.plot(x, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.xticks(x)
plt.legend(loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(x, loss, label='Training Loss')
plt.plot(x, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.xticks(x)
plt.tight_layout()
fig.savefig('train_val_plot_ResNet50.pdf', bbox_inches='tight')
plt.show()
```



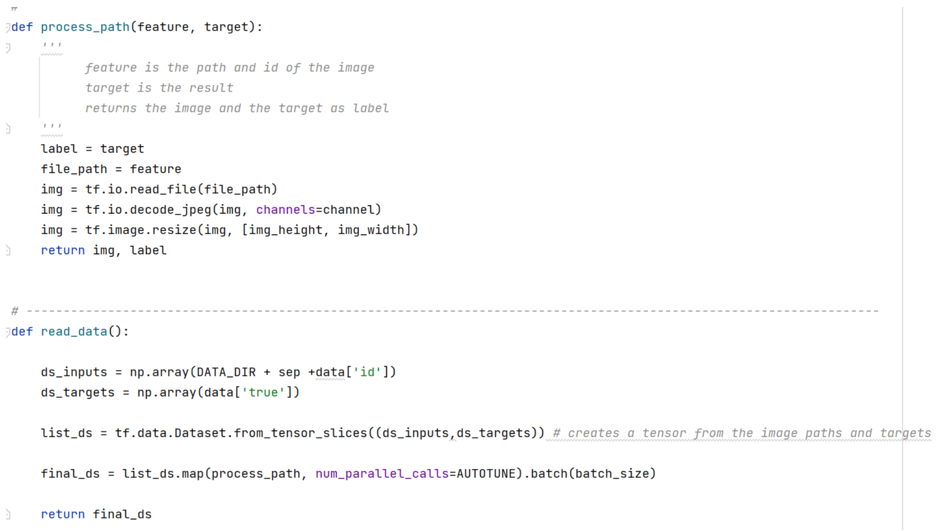
```
# ------------------------------------------------------------
#### def
# ------------------------------------------------------------
def all_data(file_dir):
    img_path = []
    img_id = []
    for root, sub_folders, files in os.walk(file_dir):
        for i in files:
            img_path.append(os.path.join(root, i))
    for i in img_path:
        a = i.split('/')[-2:]
        img_id.append(a[0] + '/' + a[1])
    labels = []
    for j in img_path:
        class_name = j.split('/')[-2]
        if class_name == 'panda':
            labels.append('panda')
        elif class_name == 'cow':
            labels.append('cow')
        elif class_name == 'spider':
            labels.append('spider')
        elif class_name == 'butterfly':
            labels.append('butterfly')
        elif class_name == 'hen':
            labels.append('hen')
        elif class_name == 'sheep':
            labels.append('sheep')
        elif class_name == 'squirrel':
            labels.append('squirrel')
        elif class_name == 'elephant':
            labels.append('elephant')
        elif class_name == 'monkey':
            labels.append('monkey')
```



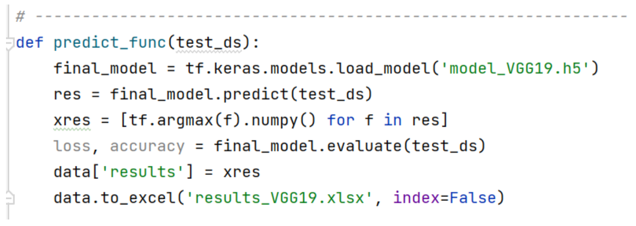
```
        elif class_name == 'cats':
            labels.append('cats')
        elif class_name == 'horse':
            labels.append('horse')
        elif class_name == 'dogs':
            labels.append('dogs')
    data = np.array([img_path, img_id, labels])
    data = data.transpose()
    path_list = list(data[:, 0])
    id_list = list(data[:, 1])
    label_list = list(data[:, 2])
    df = pd.DataFrame((path_list, id_list, label_list)).T
    df.rename(columns={0:'path', 1:"id", 2: 'target'}, inplace=True)
    return df
```



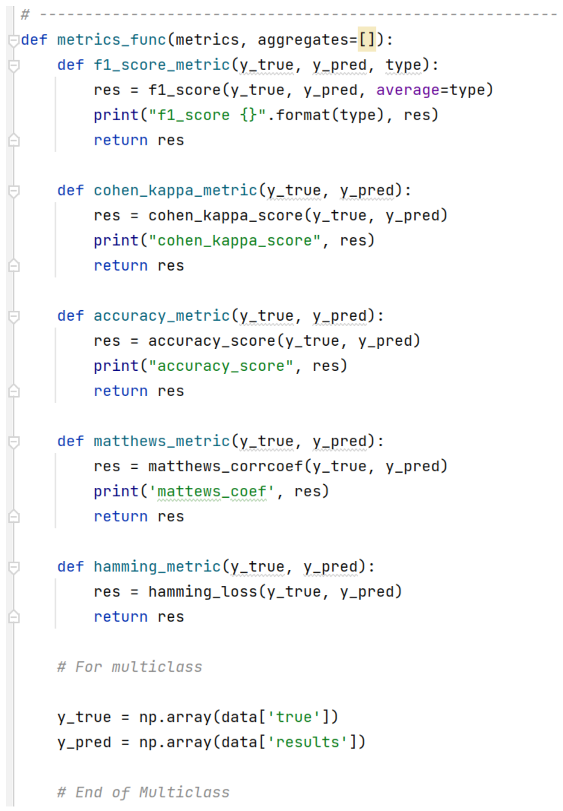
```
# -------------------------------------------------------------------------
def process_target(target_type):
    class_names = np.sort(data['target'].unique())

    if target_type == 1:

        x = lambda x: tf.argmax(x == class_names).numpy()

        final_target = data['target'].apply(x)
        data['true'] = final_target

        final_target = to_categorical(list(final_target))

        xfinal = []

        for i in range(len(final_target)):
            joined_string = ",".join(str(int(e)) for e in (final_target[i]))
            xfinal.append(joined_string)
        final_target = xfinal

        data['target_class'] = final_target
    return class_names
```



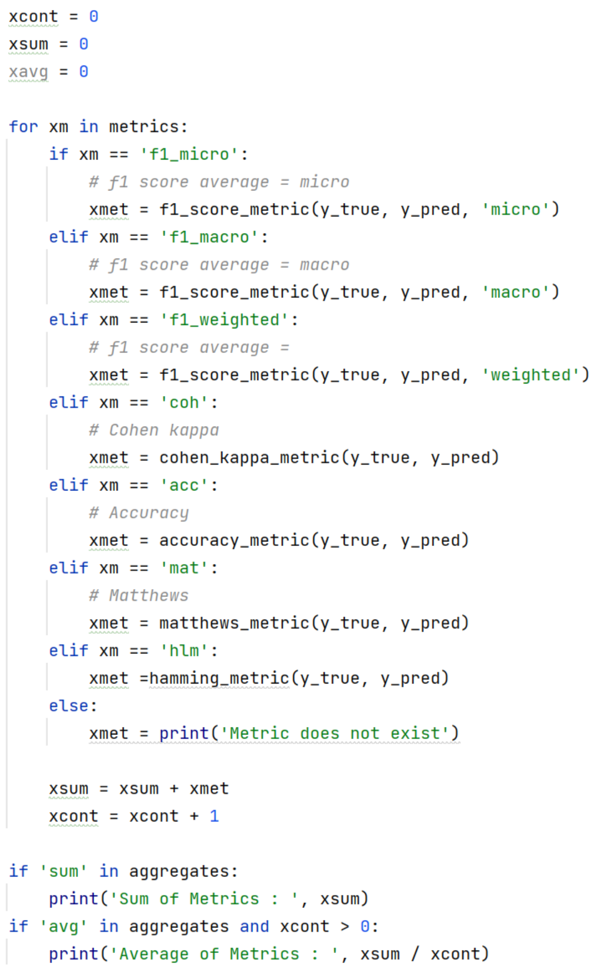
```
def process_path(feature, target):
    '''
          feature is the path and id of the image
          target is the result
          returns the image and the target as label
    '''
    label = target
    file_path = feature
    img = tf.io.read_file(file_path)
    img = tf.io.decode_jpeg(img, channels=channel)
    img = tf.image.resize(img, [img_height, img_width])
    return img, label


# -----------------------------------------------------------------------------------------------------
def read_data():

    ds_inputs = np.array(DATA_DIR + sep +data['id'])
    ds_targets = np.array(data['true'])

    list_ds = tf.data.Dataset.from_tensor_slices((ds_inputs,ds_targets)) # creates a tensor from the image paths and targets

    final_ds = list_ds.map(process_path, num_parallel_calls=AUTOTUNE).batch(batch_size)

    return final_ds
```



```
# ----------------------------------------------------------------
def predict_func(test_ds):
    final_model = tf.keras.models.load_model('model_VGG19.h5')
    res = final_model.predict(test_ds)
    xres = [tf.argmax(f).numpy() for f in res]
    loss, accuracy = final_model.evaluate(test_ds)
    data['results'] = xres
    data.to_excel('results_VGG19.xlsx', index=False)
```



```python
# ------------------------------------------------------------
def metrics_func(metrics, aggregates=[]):
    def f1_score_metric(y_true, y_pred, type):
        res = f1_score(y_true, y_pred, average=type)
        print("f1_score {}".format(type), res)
        return res

    def cohen_kappa_metric(y_true, y_pred):
        res = cohen_kappa_score(y_true, y_pred)
        print("cohen_kappa_score", res)
        return res

    def accuracy_metric(y_true, y_pred):
        res = accuracy_score(y_true, y_pred)
        print("accuracy_score", res)
        return res

    def matthews_metric(y_true, y_pred):
        res = matthews_corrcoef(y_true, y_pred)
        print('mattews_coef', res)
        return res

    def hamming_metric(y_true, y_pred):
        res = hamming_loss(y_true, y_pred)
        return res

    # For multiclass

    y_true = np.array(data['true'])
    y_pred = np.array(data['results'])

    # End of Multiclass
```



```python
xcont = 0
xsum = 0
xavg = 0

for xm in metrics:
    if xm == 'f1_micro':
        # f1 score average = micro
        xmet = f1_score_metric(y_true, y_pred, 'micro')
    elif xm == 'f1_macro':
        # f1 score average = macro
        xmet = f1_score_metric(y_true, y_pred, 'macro')
    elif xm == 'f1_weighted':
        # f1 score average =
        xmet = f1_score_metric(y_true, y_pred, 'weighted')
    elif xm == 'coh':
        # Cohen kappa
        xmet = cohen_kappa_metric(y_true, y_pred)
    elif xm == 'acc':
        # Accuracy
        xmet = accuracy_metric(y_true, y_pred)
    elif xm == 'mat':
        # Matthews
        xmet = matthews_metric(y_true, y_pred)
    elif xm == 'hlm':
        xmet =hamming_metric(y_true, y_pred)
    else:
        xmet = print('Metric does not exist')

    xsum = xsum + xmet
    xcont = xcont + 1

if 'sum' in aggregates:
    print('Sum of Metrics : ', xsum)
if 'avg' in aggregates and xcont > 0:
    print('Average of Metrics : ', xsum / xcont)
```