Міністерство освіти і науки

України Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

ЗВІТ

лабораторної роботи №6

з курсу «Програмні засоби проєктування і реалізації неромережевих систем»

Тема: «Згорткові нейронні мережі типу Xception»

Перевірив:

Шимкович В. М.

Виконав:

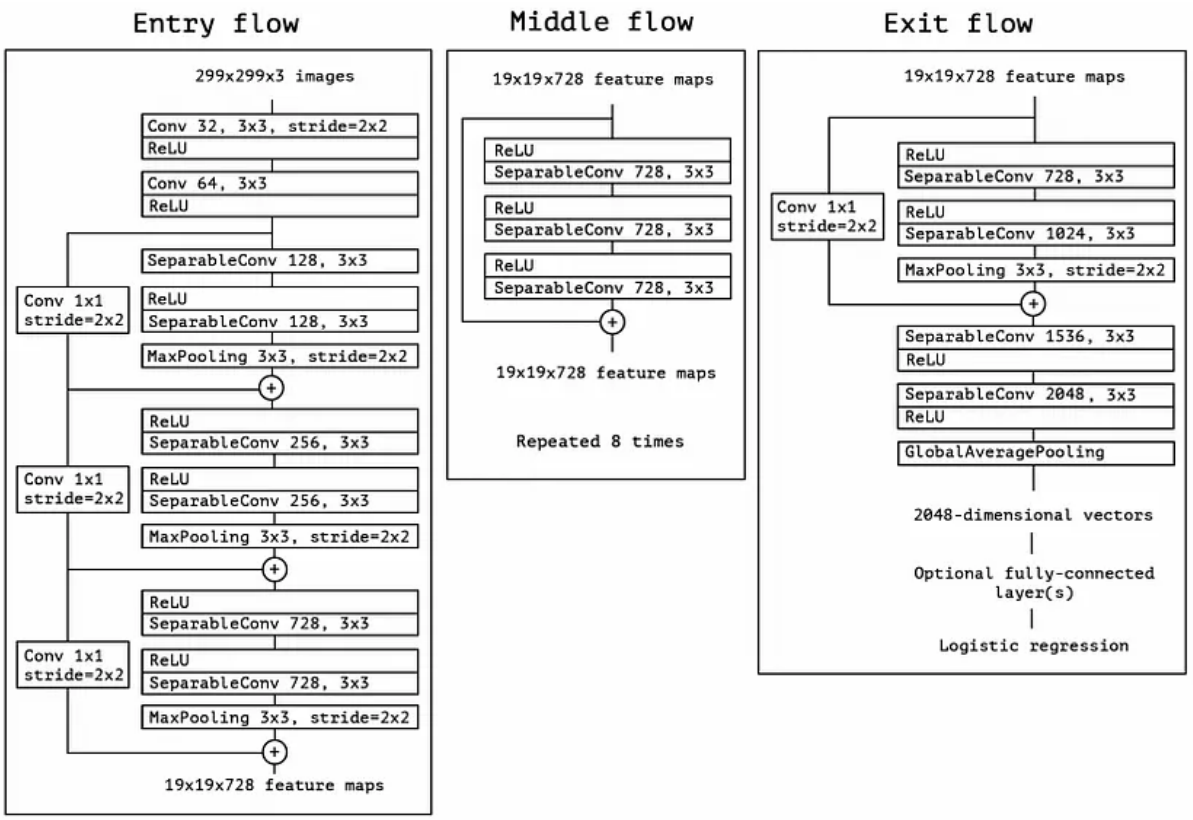
Студент Гр. ІП-01 Шпилька В.С.

Київ 2023

Завдання: Написати програму що реалізує згорткову нейронну мережу Xception для розпізнавання об’єктів на відео. Створити власний дата сет з папки на диску, навчити нейронну мережу на цьому датасеті розпізнавати логотип вашого улюбленого бренду, скажімо Apple чи BMW. Навчену нейронну мережу зберегти на комп’ютер написати програму, що відкриває та аналізує відео, результат – час на якому з’являвся логотип.

1. Реалізація нейронної мережі Xception

Архітектура мережі:



```python
def Xception(input_size, output_size):
    input_layer = tf.keras.layers.Input(shape=input_size)
    x = entry_flow(input_layer)
    x = middle_flow(x)
    x = exit_flow(x)
    if(output_size == 1):
        x = Dense(1, activation='sigmoid', name = 'output')(x)
    else:
        x = Dense(output_size, activation='softmax', name = 'output')(x)
    model = Model(input_layer, x)
    return model
```

```python
def conv_bn(x, filters, kernel_size, strides=1):
    x = Conv2D(filters=filters, kernel_size=kernel_size, strides=strides, padding='same')(x)
    x = BatchNormalization()(x)
    return x

def sep_bn(x, filters, kernel_size, strides=1):
    x = SeparableConv2D(filters=filters, kernel_size=kernel_size, strides=strides, padding='same')(x)
    x = BatchNormalization()(x)
    return x
```

Entry flow:

```python
def entry_flow(x):
    x = conv_bn(x, filters=32, kernel_size=3, strides=2)
    x = Activation('relu')(x)
    x = conv_bn(x, filters=64, kernel_size=3, strides=1)
    skip_tensor = Activation('relu')(x)

    def entry_flow_block(skip_tensor, x, filters):
        x = Activation('relu')(x)
        x = sep_bn(x, filters=filters, kernel_size=3)
        x = Activation('relu')(x)
        x = sep_bn(x, filters=filters, kernel_size=3)
        x = MaxPool2D(pool_size=3, strides=2, padding = 'same')(x)

        skip_tensor = conv_bn(skip_tensor, filters=filters, kernel_size=1, strides=2)
        x = Add()([skip_tensor, x])

        return skip_tensor, x

    skip_tensor, x = entry_flow_block(skip_tensor, x, 128)
    skip_tensor, x = entry_flow_block(skip_tensor, x, 256)
    skip_tensor, x = entry_flow_block(skip_tensor, x, 728)

    return x
```

Middle flow:

```
def middle_flow(x):
    def middle_flow_block(x):
        skip_tensor = x

        x = Activation('relu')(x)
        x = sep_bn(x, filters = 728, kernel_size = 3)
        x = Activation('relu')(x)
        x = sep_bn(x, filters = 728, kernel_size = 3)
        x = Activation('relu')(x)
        x = sep_bn(x, filters = 728, kernel_size = 3)
        x = Activation('relu')(x)

        x = Add()([skip_tensor,x])
        return x

    for _ in range(8):
        x = middle_flow_block(x)

    return x
```

Exit flow:

```
def exit_flow(x):
    skip_tensor = x

    x = Activation('relu')(x)
    x = sep_bn(x, filters=728,  kernel_size=3)
    x = Activation('relu')(x)
    x = sep_bn(x, filters=1024,  kernel_size=3)
    x = MaxPool2D(pool_size=3, strides=2, padding ='same')(x)

    skip_tensor = conv_bn(skip_tensor, filters=1024, kernel_size=1, strides=2)
    x = Add()([skip_tensor,x])

    x = sep_bn(x, filters = 1536,  kernel_size=3)
    x = Activation('relu')(x)
    x = sep_bn(x, filters = 2048,  kernel_size=3)
    x = GlobalAvgPool2D()(x)

    return x
```

Model summary занадто великий, щоб вставляти в документ. Всього нейрона мережа містить 21млн параметрів.

2. Створення датасету і тренування нейронної мережі

Під час виконання роботи було створено власний датасет. Для цього було завантажено 20 картинок основного класу та 20 картинок які не є цим клас та використано від 1 до 3 функцій зміни картинки серед яких: поворот на 90, 180, 270 градусів, відображення по горизонталі, вертикалі, зсув, зум, та комбінування з іншою картинкою.

Для створення пайплайну для даних було використано tf.data:

```
def load_data(self):
    #create and shuffle dataset
    data_dir = pathlib.Path(self.data_path)
    data_ds = tf.data.Dataset.list_files(str(data_dir/'*/*'), shuffle=False)
    previous_seed, _ = tf.compat.v1.get_seed(0)
    tf.random.set_seed(self.random_seed)
    image_count = len(data_ds)
    data_ds = data_ds.shuffle(buffer_size = image_count, seed=self.random_seed, reshuffle_each_iteration=False)
    tf.random.set_seed(previous_seed)

    #split to train, val, test
    val_size = int(image_count * self.val_percent)
    test_size = int(image_count * self.test_percent)

    train_ds = data_ds.skip(test_size + val_size)
    val_ds = data_ds.skip(test_size).take(val_size)
    test_ds = data_ds.take(test_size)

    return (train_ds, val_ds, test_ds)

def read_labels(self, path):
    with open(path, 'r') as file:
        data = file.read()

    return np.array(data.split('\n'))

def create_train_pipeline(self, ds, preprocessor):
    image_count = len(ds)
    ds = ds.shuffle(buffer_size = image_count, reshuffle_each_iteration=True).map(preprocessor.process_path, num_parallel_calls=tf.data.AUTOTUNE).batch(batch_size = self.batch_size)
    return ds

def create_test_pipeline(self, ds, preprocessor):
    ds = ds.map(preprocessor.process_path, num_parallel_calls=tf.data.AUTOTUNE).batch(batch_size = self.batch_size)
    return ds

def create_data_pipelines(self, preprocessor):
    (train_ds, val_ds, test_ds) = self.load_data()
    train_ds = self.create_train_pipeline(train_ds, preprocessor)
    val_ds = self.create_test_pipeline(val_ds, preprocessor)
    test_ds = self.create_test_pipeline(test_ds, preprocessor)
    return (train_ds, val_ds, test_ds)
```

Датасет було розділено на 3 частини для тренування, валідації та тестування. Кожну ітерацію тренувальний датасет пермішувався, розмір батча було обрано 4, а функція препроцесінгу має наступний вигляд:

```python
class LogoPreprocessing:
    def __init__(self, labels, img_height, img_width) -> None:
        self.labels = labels
        self.img_height = img_height
        self.img_width = img_width

    def get_label(self, file_path):
        parts = tf.strings.split(file_path, os.path.sep)
        one_hot = parts[-2] == self.labels
        return tf.argmax(one_hot)

    def decode_img(self, img):
        img = tf.io.decode_jpeg(img, channels=3)
        return tf.image.resize(img, [self.img_height, self.img_width])

    def process_path(self, file_path):
        label = self.get_label(file_path)
        img = tf.io.read_file(file_path)
        img = self.decode_img(img)
        return img, label

    def process_single_img(self, file_path):
        img = tf.io.read_file(file_path)
        img = self.decode_img(img)
        return img

    def preprocess_opencv_img(self, img):
        img = cv2.resize(img, (self.img_height, self.img_width))
        return img
```

Функція тренування побудована наступним чином:

Спочатку створюємо датасет

```python
logoDataset = Dataset(
    data_path=data_path,
    label_path=label_path,
    batch_size=batch_size
)

class_names = logoDataset.get_all_labels()

preprocessor = LogoPreprocessing(class_names, settings.IMAGE_HEIGHT, settings.IMAGE_WIDTH)
(train_ds, val_ds, test_ds) = logoDataset.create_data_pipelines(preprocessor)
```

Далі нейрону мережу і компілюємо її:

```
#values for schedules
initial_learning_rate = 10**(-1)
final_learning_rate = 10**(-5)
learning_rate_decay_factor = (final_learning_rate / initial_learning_rate)**(1/epochs)
steps_per_epoch = len(train_ds)

learning_rate = lr
if(lr == -1):
    learning_rate = tf.keras.optimizers.schedules.ExponentialDecay(
                        initial_learning_rate=initial_learning_rate,
                        decay_steps=steps_per_epoch,
                        decay_rate=learning_rate_decay_factor
                    )


model.compile(loss='sparse_categorical_crossentropy',
              metrics=['accuracy'],
              optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate))
```

Далі колбеки для збереження моделі і логів:

```
path_to_save = save_folder  + '/' + version + '/'

checkpoint_dir = path_to_save + "Checkpoints/"
checkpoint_path = checkpoint_dir + "cp-{epoch:04d}.ckpt"
checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             monitor='val_loss',
                             verbose=1,
                             save_weights_only = True,
                             mode='auto')

tf_path = path_to_save + "Model/tf"
fullModelSave = ModelCheckpoint(filepath=tf_path,
                             monitor='val_loss',
                             verbose=1,
                             save_best_only=True,
                             mode='auto')


log_dir = path_to_save + "Logs/"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)

callbacks_list = [checkpoint, tensorboard_callback, fullModelSave]
```

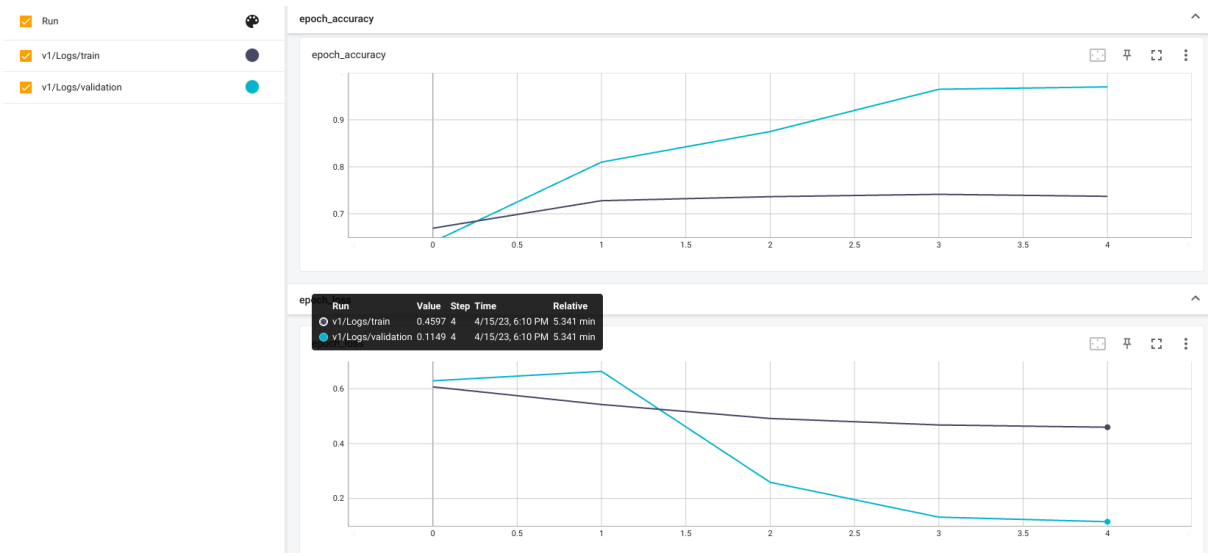
І сама функція тренування:

```
model.fit(
    train_ds,
    epochs = epochs,
    shuffle=False,
    validation_data = val_ds,
    callbacks = callbacks_list,
    verbose = 1)
```

Константи:

```
#Data
BASE_DATA_FOLDER = 'Data/MyDataset'
DATA_PATH = BASE_DATA_FOLDER + '/' + 'images'
RECOGNIZE_CLASS = 'Samsung'
OTHERS_CLASSES = 'Others'
LABELS_PATH = BASE_DATA_FOLDER + '/' + 'labels.txt'

#Gen
NUM_FILES = 20
IMAGE_KERNEL = 11
MAX_SHIFT = 20
COMBINED_SIZE = 150
MAX_OPS = 3

#Tensorflow
VAL_PERCENT = 0.1
TEST_PERCENT = 0.2
RANDOM_SEED = 42
SAVE_FOLDER = 'Artifacts/Models'
BATCH_SIZE = 2

#Model
EPOCHS = 5
DEFAULT_LR = 0.0001

#Image
IMAGE_WIDTH = 299
IMAGE_HEIGHT = 299
```

Функція втрат, точність та оцінка моделі:




```
[4]  model.evaluate(test_ds)

...  200/200 [==============================] - 55s 270ms/step - loss: 0.1155 - accuracy: 0.9800

     [0.11546775698661804, 0.9800000190734863]
```

Як бачимо модель дає дуже гарні результати, але це може бути пов’язано з невеликою вибіркою та синтетичністю в тестових даних.

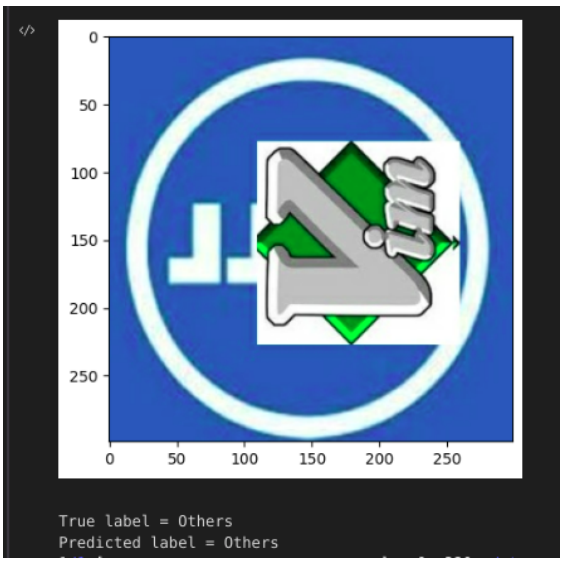
3. Завантаження моделі та її використання:

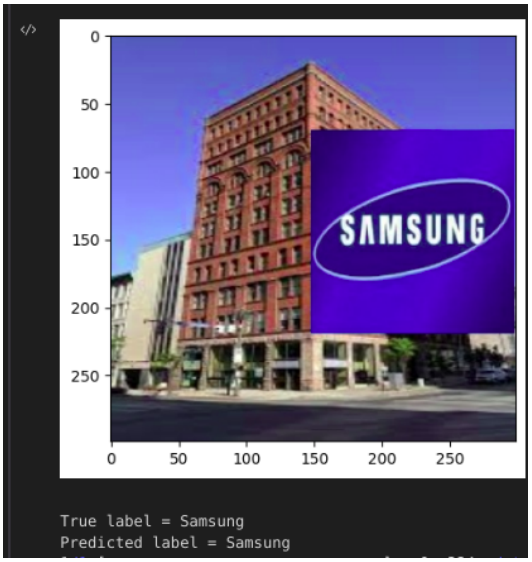
```
[3]  model = tf.keras.models.load_model("Artifacts/Models/v1/Model/tf")
     model.summary()

...  Output exceeds the size limit. Open the full output data in a text editor
     Model: "model"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #   Connected to
     =================================================================
      input_1 (InputLayer)        [(None, 299, 299, 3       0         []
                                  )]

      conv2d (Conv2D)             (None, 150, 150, 32       896       ['input_1[0][0]']
                                  )

      batch_normalization (BatchNorm  (None, 150, 150, 32   128       ['conv2d[0][0]']
      alization)                  )

      activation (Activation)     (None, 150, 150, 32       0         ['batch_normalization[0][0]']
                                  )

      conv2d_1 (Conv2D)           (None, 150, 150, 64       18496     ['activation[0][0]']
                                  )

      batch_normalization_1 (BatchNo  (None, 150, 150, 64   256       ['conv2d_1[0][0]']
      rmalization)                )

      activation_2 (Activation)   (None, 150, 150, 64       0         ['batch_normalization_1[0][0]']
                                  )

     ...
     Total params: 20,890,793
     Trainable params: 20,836,265
     Non-trainable params: 54,528
     _________________________________________________________________
```

```python
for images, true_classes in test_ds.take(10):
    predictions = model.predict(images)
    true_class = true_classes.numpy()
    pred_class = np.round(predictions)
    for index, img in enumerate(images):
        img = img.numpy()
        imgplot = plt.imshow(images[index].numpy().astype('uint8'))
        plt.show()
        print('True label =', class_names[true_class[index]])
        print('Predicted label =', class_names[int(np.round(pred_class[index]))])
        if(index > 10):
            break
```

[25]

Декілька картинок з тестового датасету:





Обробка відео:

```python
video = cv2.VideoCapture('Data/samsung.mp4')
frame_no = 0
last_predictions = []
last_result = False
begin_time = []
end_time = []
while video.isOpened():
    ret, frame = video.read()
    if(ret):
        frame_no += 1
        if(frame_no % 4 != 0):
            continue

        process_frame = preprocessor.preprocess_opencv_img(frame)
        process_frame = np.expand_dims(process_frame, axis = 0)
        predictions = model.predict(process_frame, verbose = 0)
        pred_class = predictions[0]
        last_predictions.append(pred_class)
        if(len(last_predictions) > 3):
            last_predictions.pop(0)
        avg_prediction = sum(last_predictions) / len(last_predictions)
        if(avg_prediction > 0.5 and not last_result):
            last_result = True
            time = video.get(cv2.CAP_PROP_POS_MSEC) / 1000
            print("Start =", time)
            begin_time.append(time)

        if(avg_prediction < 0.5 and last_result):
            last_result = False
            time = video.get(cv2.CAP_PROP_POS_MSEC) / 1000
            print("End =", time)
            begin_time.append(time)

        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            break
    else:
        cv2.destroyAllWindows()
        break
```

[5] ✓ 24.0s

```
Start = 7.433333333333334
End = 15.833333333333334
```

Висновок: В результаті виконання лабораторної роботи було побудовано згорткову нейрону мережу Xception для бінарної класифікації чи містить картинка бренд Samsung. Всього нейрона мережа має 21млн параметрів. Для даного датасету дана нейрона мережа показала гарні результати, а саме точність в 98 відсотків. Дана точність обумовлена невеликим і синтетичним датасетом, але незважаючи на це, дана нейронка гарно підходить навіть для обробки відео і класифікації фреймів з нього.