EE5353 Program Assignment 9

Transfer Learning using Google Colab

Transfer learning make use of the knowledge gained while solving one problem and applying it to a different but related problem.

In this assignment, we will use a pretrained network and apply it to a new dataset. There are a lot of different pretrained networks in keras.

Below is the link for different pretrained networks we can use - https://keras.io/applications/

We will be using VGG16

This assignment consists of 2 parts (training, validation and testing data are different from each other)

1st create a CNN as we did in the previous assignments with below given parameters. Prediction code should be included.

2nd run transfer learning python file provided in the link and write the prediction code.

Data information – 4 classes (Humans, Horses, cats and dogs)

https://drive.google.com/drive/folders/1CkWPAlaf5ceCKodqYPXZNiDKrlRWnSpw?usp=sharing

Files – train with 4 classes, Validation with 4 classes, Testing with all images

I AM INCLUDING A NEW TYPE OF IMAGE AUGMENTATION. YOU JUST NEED TO INPUT THE LOCATION OF THE FILE AND IT WILL DIRECTLY USE IMAGES FROM THE DIRECTORY. Image size is 256 x 256. So there is no need to input images in this assignment as he we did before (code is at the end of the document)

Task 1 (independent of Task 2) (training, validation and testing data is independent)

Create a CNN It should have the following layers

- Convolutional layer with 128 filters, Size of the filters is 3, 3, stides = 1 and relu activation
- Convolutional layer with 128 filters, Size of the filters is 3, 3, stides = 1 and relu activation
- Pooling layer with pool size 3,3
- Convolutional layer with 256 filters, Size of the filters is 3, 3, stides = 1 and relu activation
- Convolutional layer with 256 filters, Size of the filters is 3, 3, stides = 1 and relu activation
- Flattening
- Dense layer fully connected with 64 hidden units and relu activations
- Dropout layer with rate 0.5
- Dense layer fully connected with 64 hidden units and relu activations
- Dropout layer with rate 0.5
- Final dense fully connected layer with number of classes and softmax activation.

For Final testing just use model.predict .**It should print the class number. And show if the class predicted is correct class as a code not manual**

****Note – if you don't have enough ram the code will crash**

Task 2 (training code is present) (training, validation and testing data is independent)

transfer learning training code will be provided. You just need to implement the prediction code. Similar to above.

VGG16_transferlearning.ipynb code is in the same link as the image data

Code (**train_dir – training file location, validation_dir – validation file location)**

Below code should be inserted after creating a model

```
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Change the batchsize according to your system RAM
train_batchsize = 50
val_batchsize = 10

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(image_size, image_size),
```

```
        batch_size=val_batchsize,
        class_mode='categorical',
        shuffle=False)


# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
# Train the model
history = model.fit_generator(
      train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size ,
      epochs=5,
      validation_data=validation_generator,
      validation_steps=validation_generator.samples/validation_generator.batc
h_size,
      verbose=1)
```