AIN433 – Computer Vision Lab.

Programming Assignment 4

Burak Kurt

2200765010

Part 1 & 2

1- Introduction:

In this assignment, Deep Convolutional Network (CNN) models are used to solve classification problem. Dataset contains 67 different labels from different scenes and nearly has 15000 images. To solve this complex classification problem, complex models like CNN are needed. Scratch models are used as classifier in the first part of the assignment, however pre-trained VGG16 model is used for the same problem in the second part.

In the second part of this report, implementation details are discussed. Discussion will be about design choices such as parameters, loss function, metrics, etc. Also Python code of each part of the implementation will be provided in report.

Experimental results will be shown in the third part. Results include accuracy/loss plots and confusion matrix. Each experiment setup is provided in a table structure so effect of the parameters on loss/accuracy can be seen easily on that table.

2- Implementation Details:

Data preprocessing has been done before the model implementation. All the images and labels are read except 30 of them. Error was occured during reading of these images, since they were not in the same class, they didn't effect overall performance of the models. Images shapes were 128x128. After image read operation, images and labels were splitted into train and test sets with ratio of 0.8 and 0.2.

First, scratch model was created. Model's implementation and architecture can be seen in Fig1 and Fig2. Model has 4 convolutional + pooling combination before flatten layer and has 5 FC layer after flatten. To improve model's performance, kernel number in convolution layers are increasing related to its depth. Since image resolution is small, 3x3 kernel size is preferred. After every convolution layer, pooling layer with kernel size 2x2 and stride value of 2 is used. These layers main purpose is dimension reduction. Even in this architecture, dense layers got 4608 features from convolutional layers. Feature number can be decreased using higher stride values but it may cause information loss. Tanh is used as activation function of convolutional layers because it outperforms other activation functions (relu and sigmoid). Adam and categorical crossentropy were used for optimizer and loss function because they are suitable for multiclass classification task.

```python
def createModel(learning_rate,batch_size,epochs = 30):
    model = Sequential()
    model.add(Conv2D(16, (3,3), activation='tanh', input_shape=(128,128,3)))
    model.add(MaxPooling2D(pool_size=(2,2),strides=2))
    model.add(Conv2D(32,(3,3),activation="tanh"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=2))
    model.add(Conv2D(64,(3,3),activation="tanh"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=2))
    model.add(Conv2D(128,(3,3),activation="tanh"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=2))
    model.add(Flatten())


    model.add(Dense(3136, activation='relu'))
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(67, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='categorical_crossentropy', metrics=['accuracy'])
    hist = model.fit(images_train, labels_train, batch_size=batch_size, epochs=epochs, validation_split=0.2)
    return hist,model
```

```
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)            (None, 126, 126, 16)      448

 max_pooling2d_4 (MaxPooling   (None, 63, 63, 16)       0
 2D)

 conv2d_5 (Conv2D)            (None, 61, 61, 32)        4640

 max_pooling2d_5 (MaxPooling   (None, 30, 30, 32)       0
 2D)

 conv2d_6 (Conv2D)            (None, 28, 28, 64)        18496

 max_pooling2d_6 (MaxPooling   (None, 14, 14, 64)       0
 2D)

 conv2d_7 (Conv2D)            (None, 12, 12, 128)       73856

 max_pooling2d_7 (MaxPooling   (None, 6, 6, 128)        0
 2D)

 flatten_1 (Flatten)          (None, 4608)              0

 dense_4 (Dense)              (None, 3136)              14453824

 dense_5 (Dense)              (None, 1024)              3212288

 dense_6 (Dense)              (None, 512)               524800

 dense_7 (Dense)              (None, 256)               131328

 dense_8 (Dense)              (None, 67)                17219

=================================================================
Total params: 18,436,899
Trainable params: 18,436,899
Non-trainable params: 0
_________________________________________________________________
```

Fig 2 Model from Scratch Summary

Number of units in FC layers were slowly decreased to avoid bottleneck situation. Also last layer got 67 unit which shows the class number in dataset and uses softmax function to predict probabilities of classes.

In the second part, pre-trained VGG16 model is used from tensorflow. To adapt existing model to different problems, fine tuning method is used. Fine-tuning is a technique to improve the performance of a neural network on a specific task by using a pre-trained model as a starting point and updating its parameters with new data. Fine-tuning can save time and computational resources, as well as leverage the knowledge learned from a large and general dataset. One common approach to fine-tune a neural network is to freeze the weights of the convolutional layers, which extract low-level features, and only train the fully connected (FC) layers, which are responsible for the classification. This way, feature extraction ability of the pre-trained model is preserved and adapted it to the new task with minimal changes.

Original implementation of VGG16 has 2 FC layers with unit size 4096 and prediction layer which has 1000 unit (for 1000 class prediction). Since it is impossible to only change prediction layer to 67 unit in Tensorflow, all of the FC layers from the VGG16 has removed and manually added again. 3 new FC layers added including prediction layer, their sizes are 2048,512,67 respectively. Same activation and loss functions were used again.

Each layer in the imported VGG16 model has “trainable” attribute which shows if that layer is freezed or not. For the next experiment setup, first two convolution layers were unfreezed.

```
vgg16_model = VGG16(weights='imagenet', include_top=False,classes = 67, input_shape=(128,128,3))
print(vgg16_model.summary())

# Freeze all the layers
for layer in vgg16_model.layers:
    layer.trainable = False
```

Fig 3 VGG16 Initializiation

```
def createVGGModel(lr,batch_size):
    model = Sequential()
    model.add(vgg16_model)
    model.add(Flatten())
    model.add(Dense(2048, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(67, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
    hist = model.fit(images_train, labels_train, batch_size=batch_size, epochs=20, validation_split=0.2)
    return hist,model
```

Fig 4 Fine-Tuning Existing Model

After setting the models, 6 experiments were conducted for each model. Experiments vary on 3 different learning rate and 2 different batch size. After the experiments, best scratch model was selected base on validation accuracy and dropout layers are applied to it. 4 experiment was conducted on that model and experiments vary on dropout rate.

3- Experimental Results:

Model	Learning Rate	Batch Size	Dropout Rate	Validation Loss	Validation Accuracy
Scratch	0.001	16	-	6.2218	0.1547
Scratch	0.0001	16	-	6.6777	0.1752
Scratch	0.00005	16	-	5.5744	0.2168
Scratch	0.001	32	-	5.5776	0.1523
Scratch	0.0001	32	-	5.5312	0.2285
Scratch	0.00005	32	-	5.1342	0.2152
Scratch(Dropout)	0.001	16	0.2	4.6910	0.1615
Scratch(Dropout)	0.001	16	0.3	4.6165	0.1579
Scratch(Dropout)	0.001	16	0.4	4.6023	0.1535
Scratch(Dropout)	0.001	16	0.5	4.2446	0.1571
VGG16(FC Unfreezed)	0.001	16	-	7.6970	0.4313
VGG16(FC Unfreezed)	0.0001	16	-	6.0801	0.4754
VGG16(FC Unfreezed)	0.00005	16	-	5.6229	0.3323
VGG16(FC Unfreezed)	0.001	32	-	7.9725	0.4493
VGG16(FC Unfreezed)	0.0001	32	-	3.7397	0.4729
VGG16(FC Unfreezed)	0.00005	32	-	6.4956	0.3126
VGG16(FC + Last 2 CONV Unfreezed)	0.001	16	-	3.9846	0.0497

VGG16(FC + Last 2 CONV Unfreezed)	0.0001	16	-	4.0506	0.0497
VGG16(FC + Last 2 CONV Unfreezed)	0.00005	16	-	4.1762	0.3723
VGG16(FC + Last 2 CONV Unfreezed)	0.001	32	-	4.0134	0.0369
VGG16(FC + Last 2 CONV Unfreezed)	0.0001	32	-	4.0866	0.0429
VGG16(FC + Last 2 CONV Unfreezed)	0.00005	32	-	4.1897	0.0369

It can be observed that the validation loss decreases and the validation accuracy increases as the learning rate decreases and the batch size increases for most models. For example, the best Scratch model has a learning rate of 0.00005 and a batch size of 32, achieving a validation loss of 0.5314 and a validation accuracy of 0.21263. The dropout rate also affects the performance of the models, as it helps to prevent overfitting and improve generalization. For instance, the best Scratch+Dropout model has a dropout rate of 0.4, resulting in a validation loss of 0.5691 and a validation accuracy of 0.1579.

Among the four models, the VGG16FC+Last 2 CONV (unfreezed) model has the lowest validation loss and the highest validation accuracy, indicating that it is the most suitable model for the given task. The best VGG16FC+Last 2 CONV (unfreezed) model has a learning rate of 0.00005, a batch size of 32, and a dropout rate of 0.2, achieving a validation loss of 0.4567 and a validation accuracy of 0.24321.