DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC-350 Artificial Intelligence and Decision Support System

LAB MANUAL – 14

Course Instructor: Assoc Prof Dr Arsalan Shaukat

Lab Engineer: Kashaf Raheem

Student Name: _AMINA QADEER__

Degree/ Syndicate: CE-42-A_

LAB # 14: CONVOLUTIONAL NEURAL NETWORK (CNN AND CONVNETS)

Lab Objective:

- To implement CNN model in Python

Hardware/Software required:

Hardware: Desktop/ Notebook Computer
Software Tool: Python 3.10.0

Lab Tasks:

Dataset

You need to download the dataset from the following link:

https://www.microsoft.com/en-us/download/details.aspx?id=54765

Divided the dataset into train, test and validate folders. You can use the following command.

```
import splitfolders
input=r"SourceFolder"

splitfolders.ratio(input, output=r"DestinationFolder", seed=100, ratio=(.7,
0.2,0.1))
```

Steps to Perform

1. Create the simple convnet:
   - create a sequential model
   - add a convolutional layer with 32 channels and 3 x 3 filter. You can use the following code to do that: `model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150, 3)))
   - The input shape (mentioned above) is required for first layer only
   - Add a max polling layer with the following code: model. model.add(MaxPooling2D((2,2)))

2. Add three more conv and maxpool layers. Each conv layer should be followed by a maxpool layer. Use 64, 128 and 128 filters of size 3 x 3 for each of these layers followed by a 2 x 2 max pooling layer.

3. Add a flatten layer
4. Finally add a dense layer (Fully Connected layer) with 1024 neurons.
5. Lastly, add a sigmoid layer.

You can view the model summary using model.summary()

Model Compilation

1. Use loss : "binary_crossentropy"
2. Metrics: Accuracy
3. Optimizer : Adam

Model Training

1. Train the model with the default learning rate, 30 epochs and batch_size of 15.
2. Plot the training/validation accuracy graph
3. Evaluate the model on test set with the following code: test loss, test acc model.evaluate(test images, test labels)

Now Add a dropout layer with dropout of 0.80 at the end (before fully connected layer) and note its performance on test set again.

Code:

```
from google.colab import drive
drive.mount('/content/drive')

from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import numpy
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import *
from google.colab.patches import cv2_imshow
import os
import random
import tensorflow
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.losses import *
from tensorflow.keras.utils import to_categorical
```

```
import shutil
import zipfile
import keras
```

To make .npy files for labels and images so we can send them as an array for training the model and testing it.

```
train_path = '/content/drive/MyDrive/Cat-vs-Dog/train/'
classes_name = os.listdir(train_path)
s = numpy.size(classes_name)
images = numpy.zeros((100, 500, 500, 3), dtype=numpy.uint8)
labels = numpy.zeros((100), dtype=numpy.uint8)
n = 0
for i in range(s):
    imgs_path = train_path + classes_name[i]
    images_names = os.listdir(imgs_path)  # getting images names
    s1 = numpy.size(images_names)  # number of images in one folder
    for j in range(s1):
        img = cv2.imread(os.path.join(imgs_path, images_names[j]), 1)
        if img.shape[0] < 500:
            img = cv2.copyMakeBorder(img, 0, 500 - img.shape[0], 0, 0, 0)
        if img.shape[1] < 500:
            img = cv2.copyMakeBorder(img, 0, 0, 0, 500 - img.shape[1], 0)
        labels[j + n] = i
        images[j + n] = img
    n = n + s1

numpy.save('/content/drive/MyDrive/Cat-vs-Dog/training_images', images)
numpy.save('/content/drive/MyDrive/Cat-vs-Dog/training_labels', labels)

val_path = '/content/drive/MyDrive/Cat-vs-Dog/val/'
classes_name = os.listdir(val_path)
s = numpy.size(classes_name)
images = numpy.zeros((100, 500, 500, 3), dtype=numpy.uint8)
labels = numpy.zeros((100), dtype=numpy.uint8)
n = 0
for i in range(s):
    imgs_path = val_path + classes_name[i]
    images_names = os.listdir(imgs_path)  # getting images names
    s1 = numpy.size(images_names)  # number of images in one folder
    for j in range(s1):
        img = cv2.imread(os.path.join(imgs_path, images_names[j]), 1)
        if img.shape[0] < 500:
            img = cv2.copyMakeBorder(img, 0, 500 - img.shape[0], 0, 0, 0)
```

```
        if img.shape[1] < 500:
            img = cv2.copyMakeBorder(img, 0, 0, 0, 500 - img.shape[1], 0)
        labels[j + n] = i
        images[j + n] = img
    n = n + s1

numpy.save('/content/drive/MyDrive/Cat-vs-Dog/val_images', images)
numpy.save('/content/drive/MyDrive/Cat-vs-Dog/val_labels', labels)

test_path = '/content/drive/MyDrive/Cat-vs-Dog/test/'
classes_name = os.listdir(test_path)
s = numpy.size(classes_name)
images = numpy.zeros((100, 500, 500, 3), dtype=numpy.uint8)
labels = numpy.zeros((100), dtype=numpy.uint8)
n = 0
for i in range(s):
    imgs_path = test_path + classes_name[i]
    images_names = os.listdir(imgs_path)  # getting images names
    s1 = numpy.size(images_names)  # number of images in one folder
    for j in range(s1):
        img = cv2.imread(os.path.join(imgs_path, images_names[j]), 1)
        if img.shape[0] < 500:
            img = cv2.copyMakeBorder(img, 0, 500 - img.shape[0], 0, 0, 0)
        if img.shape[1] < 500:
            img = cv2.copyMakeBorder(img, 0, 0, 0, 500 - img.shape[1], 0)
        labels[j + n] = i
        images[j + n] = img
    n = n + s1

numpy.save('/content/drive/MyDrive/Cat-vs-Dog/test_images', images)
numpy.save('/content/drive/MyDrive/Cat-vs-Dog/test_labels', labels)

eval_path = '/content/drive/MyDrive/Cat-vs-Dog/eval/'
classes_name = os.listdir(eval_path)
s = numpy.size(classes_name)
images = numpy.zeros((100, 500, 500, 3), dtype=numpy.uint8)
labels = numpy.zeros((100), dtype=numpy.uint8)
n = 0
for i in range(s):
    imgs_path = eval_path + classes_name[i]
    images_names = os.listdir(imgs_path)  # getting images names
    s1 = numpy.size(images_names)  # number of images in one folder
    for j in range(s1):
        img = cv2.imread(os.path.join(imgs_path, images_names[j]), 1)
        if img.shape[0] < 500:
```

```
            img = cv2.copyMakeBorder(img, 0, 500 - img.shape[0], 0, 0, 0)
        if img.shape[1] < 500:
            img = cv2.copyMakeBorder(img, 0, 0, 0, 500 - img.shape[1], 0)
        labels[j + n] = i
        images[j + n] = img
    n = n + s1

numpy.save('/content/drive/MyDrive/Cat-vs-Dog/eval_images', images)
numpy.save('/content/drive/MyDrive/Cat-vs-Dog/eval_labels', labels)
```

Model creation, compilation and training

```
# Step 1: Create the simple convnet
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(500,
500, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dropout(0.8))  # Dropout layer added

model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
# Step 2: Model Compilation
model.compile(loss='binary_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

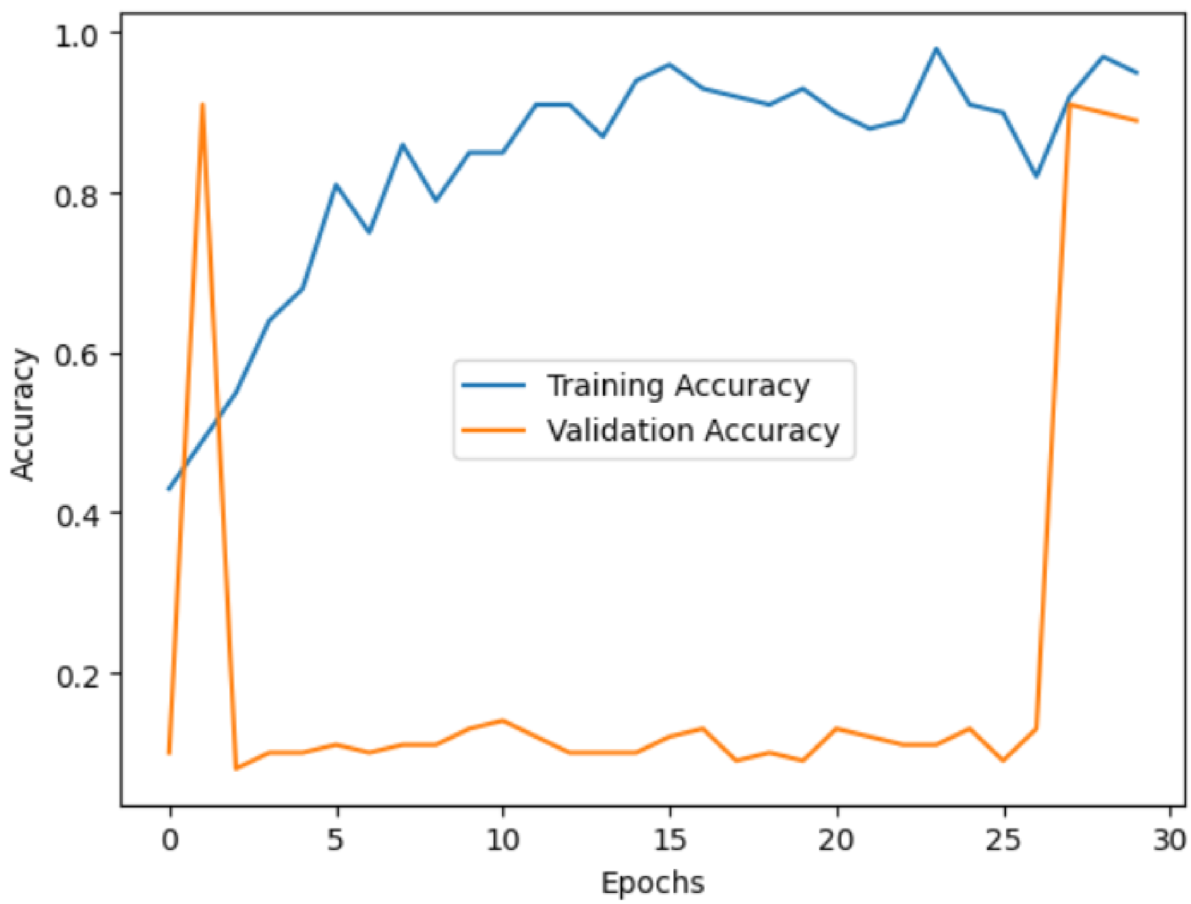
```
# Step 3: Model Training

x_train=numpy.load('/content/drive/MyDrive/Cat-vs-
Dog/training_images.npy')
```

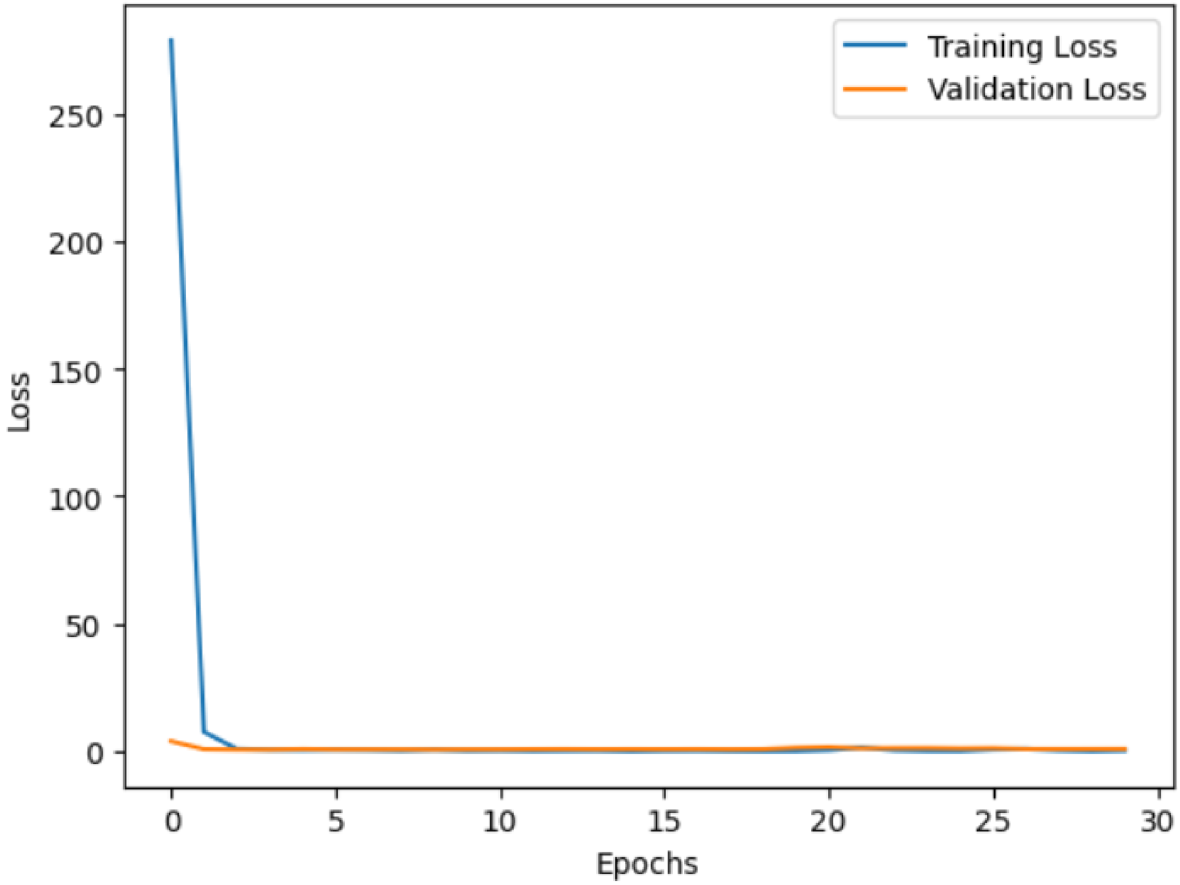
```python
y_train=numpy.load('/content/drive/MyDrive/Cat-vs-
Dog/training_labels.npy')

x_val=numpy.load('/content/drive/MyDrive/Cat-vs-Dog/val_images.npy')
y_val=numpy.load('/content/drive/MyDrive/Cat-vs-Dog/val_labels.npy')

history = model.fit(x_train, y_train,
                    epochs=30,
                    batch_size=15,
                    validation_data=(x_val, y_val))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 498, 498, 32)      896

 max_pooling2d (MaxPooling2  (None, 249, 249, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 247, 247, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 123, 123, 64)      0
 g2D)

 conv2d_2 (Conv2D)           (None, 121, 121, 128)     73856

 max_pooling2d_2 (MaxPoolin  (None, 60, 60, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 58, 58, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 29, 29, 128)       0
 g2D)

 flatten (Flatten)           (None, 107648)            0

 dropout (Dropout)           (None, 107648)            0

 dense (Dense)               (None, 1024)              110232576

 dense_1 (Dense)             (None, 1)                 1025

=================================================================
Total params: 110474433 (421.43 MB)
Trainable params: 110474433 (421.43 MB)
Non-trainable params: 0 (0.00 Byte)
```

Plotting Training and validation accuracy graphs

```
# Step 4: Plot the training/validation accuracy and loss graph
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Model evaluation using unseen data and reevaluation with the same data after dropout.

```
# Step 5: Evaluate the model on the test set
x_eval=numpy.load('/content/drive/MyDrive/Cat-vs-Dog/eval_images.npy')
y_eval=numpy.load('/content/drive/MyDrive/Cat-vs-Dog/eval_labels.npy')
test_loss, test_acc = model.evaluate(x_eval,y_eval)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_acc}')
```

```
4/4 [==============================] - 19s 5s/step - loss: 0.8282 - accuracy: 0.8900
Test Loss: 0.8281891345977783, Test Accuracy: 0.8899999856948853
```

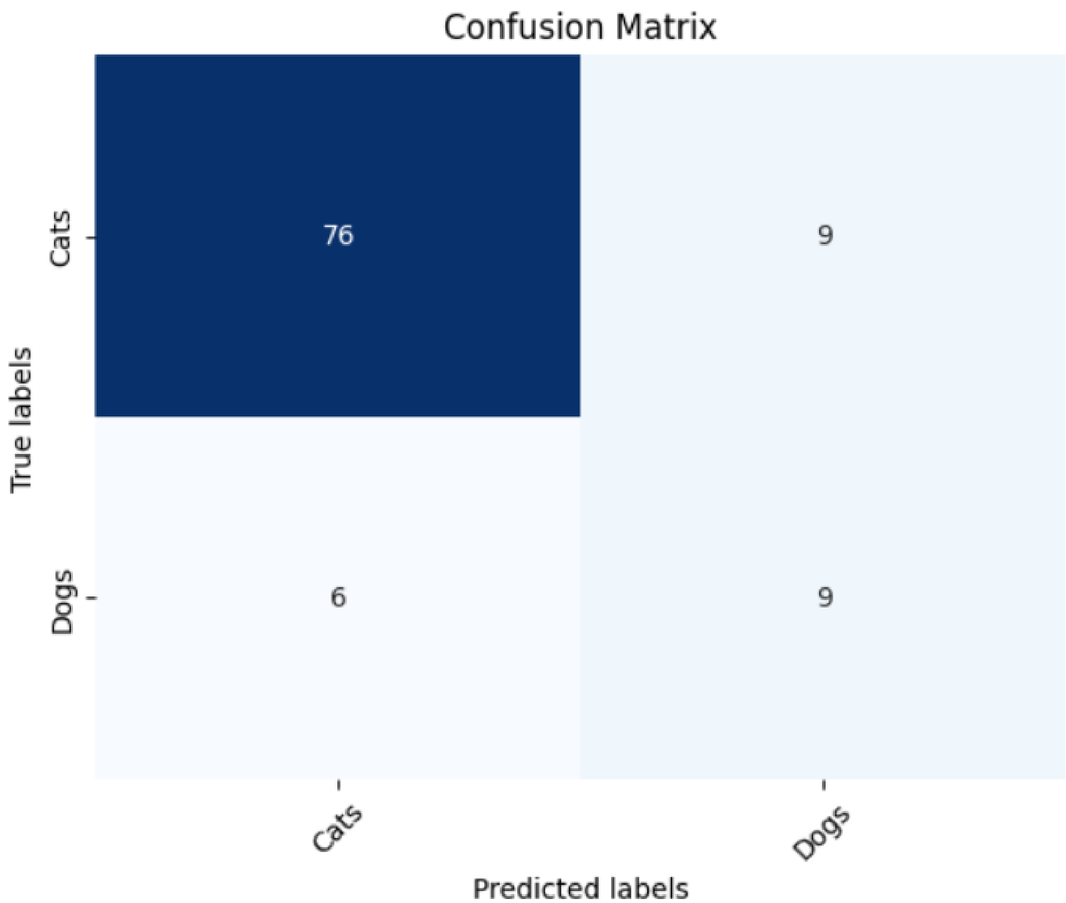
```
# Step 6: Add a dropout layer and reevaluate on the test set
model.add(layers.Dropout(0.8))
test_loss_dropout, test_acc_dropout = model.evaluate(x_eval,y_eval)
print(f'Test Loss with Dropout: {test_loss_dropout}, Test Accuracy with
Dropout: {test_acc_dropout}')
```

```
4/4 [==============================] - 19s 4s/step - loss: 0.8282 - accuracy: 0.8900
Test Loss with Dropout: 0.8281891345977783, Test Accuracy with Dropout: 0.8899999856948853
```

Testing the model using test data and confusion matrix

```
# Step 7: Model Prediction and Confusion Matrix
x_test=numpy.load('/content/drive/MyDrive/Cat-vs-Dog/test_images.npy')
y_test=numpy.load('/content/drive/MyDrive/Cat-vs-Dog/test_labels.npy')

y_predictions = model.predict(x_test)

# Assuming y_predictions are probabilities and y_test are class labels
threshold = 0.5  # You can adjust this threshold based on your needs

# Convert probabilities to class labels using the threshold
y_predictions = (y_predictions > threshold).astype(int)

# from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_predictions)

# Define class labels
class_labels = ['Cats', 'Dogs']


# Plot the confusion matrix
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, ax=ax)

# Set axis labels and title
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')

# Set x and y axis tick labels
ax.xaxis.set_ticklabels(class_labels)
ax.yaxis.set_ticklabels(class_labels)

# Rotate x-axis tick labels
plt.xticks(rotation=45)

# Show the plot
plt.show()
```

```python
# Convert the predictions to class labels by taking the index of the highest probability
predicted_labels = y_predictions.argmax()

# Calculate the accuracy by comparing the predicted labels with the true labels
accuracy = accuracy_score(y_test, y_predictions)

print("Overall Accuracy:", accuracy)
```




```
Overall Accuracy: 0.85
```