

## Assignment No. 2

**Aim:** Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report.

- Develop an application and test the user given inputs.

Analyze the result with and without regularization/dropout

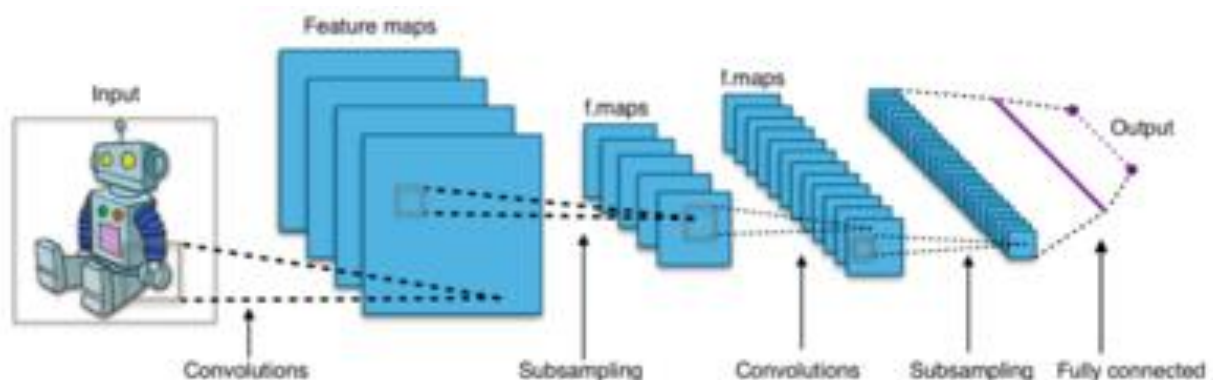
### Objectives:

1. To learn about classification
2. To learn CNN
3. To demonstrate and analyse the results

### Theory:

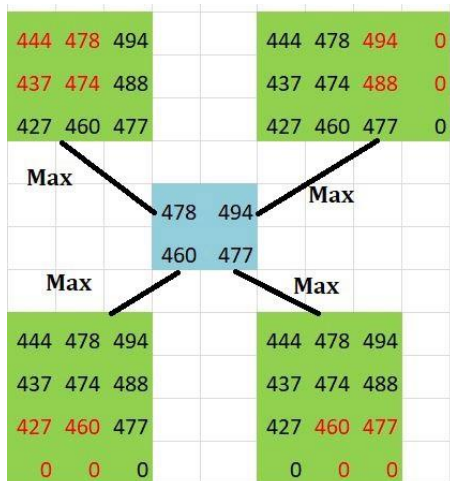
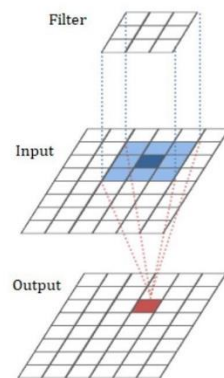
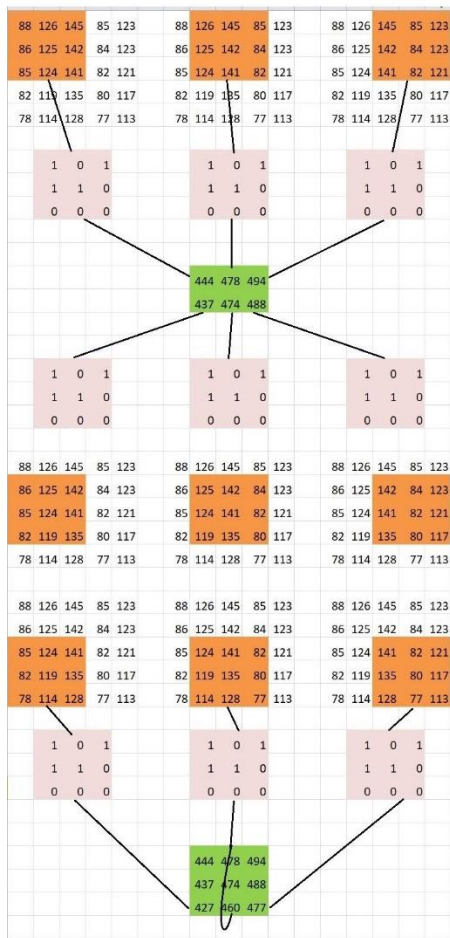
A **convolutional neural network (CNN, or ConvNet)** is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as **shift invariant** or **space invariant artificial neural networks (SIANN)**, based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.



# Mathematics

## Convolution Process



## Batch Normalization

**Batch normalization** is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-**batch**. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

## Regularisation

This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. A simple relation for linear regression looks like this. Here  $Y$  represents the learned relation and  $\beta$  represents the coefficient estimates for different variables or predictors( $X$ ).

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

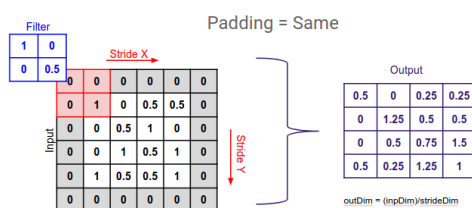
The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$RSS = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

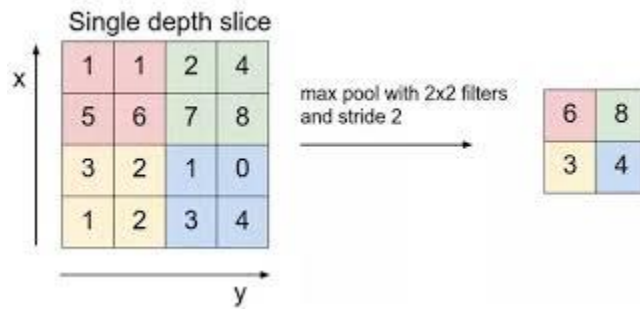
## Padding

**Padding** is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a **CNN**. For example, if the **padding** in a **CNN** is set to zero, then every pixel value that is added will be of value zero.



## Strides

**Stride** is the number of pixels shifts over the input matrix. When the **stride** is 1 then we move the filters to 1 pixel at a time. When the **stride** is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a **stride** of 2.



Code:

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

downloaded = drive.CreateFile({'id':"1NwrithRqi1lcpAPadM5Rz0AQrqmr-2DD"})
downloaded.GetContentFile('CatvsDogs.rar')

!unrar x "/content/CatvsDogs.rar" "/content/"

import pandas as pd
import tensorflow as tf
from tensorflow.keras import models, Sequential, layers, preprocessing
import os

file_names=os.listdir("/content/train")
dogorcat=[]
for name in file_names:
    category=name.split('.')[0]
    if category=='dog':
        dogorcat.append("DOG")
    else:
        dogorcat.append("CAT")
train=pd.DataFrame({
    'filename':file_names,
    'category':dogorcat
})

model=models.Sequential()
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(64,64,3)))
#model.add(layers.Conv2D(32,(3,3),activation='relu'))
model.add(layers.BatchNormalization())
```

```

model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.BatchNormalization())
#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.BatchNormalization())
#model.add(layers.Conv2D(128,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Flatten())
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(256,activation="relu"))
model.add(layers.Dense(128,activation="relu"))
model.add(layers.Dense(2,activation="softmax"))

model.compile(optimizer="adam",loss='categorical_crossentropy',metrics=['accuracy'])

training = preprocessing.image.ImageDataGenerator(rotation_range=15, rescale=1
./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1)
validation=preprocessing.image.ImageDataGenerator(rotation_range=15, rescale=1
./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1)

trainingdata = training.flow_from_dataframe(train,"/content/train",x_col='file
name',y_col='category',target_size=(64,64),class_mode='categorical')

model.fit(trainingdata,epochs=10)

from google.colab import drive
drive.mount('/content/drive')
os.makedirs("drive/My Drive/Models",exist_ok=True)
model.save("drive/My Drive/Models")

```

## GUI

```

from PIL import Image,ImageOps
import os
import streamlit as st
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras.preprocessing import image
import numpy as np
st.set_option('deprecation.showfileUploaderEncoding', False)
def load_models(img):
    model = models.load_model('D:/Programs/ML1/ML2/Assignment3/')

```

```

image=img.resize((64,64))
image_array=np.array(image)
#image_array=tf.image.rgb_to_grayscale(image_array)
image_array=(tf.reshape(image_array,(image_array.shape[0],image_array.shap
e[0],3)))/255
image_array=np.array([image_array])

prediction = model.predict_classes(image_array)
return prediction

def upload_images():
    uploaded_file = st.file_uploader("Choose an Image ...", type="jpg")
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded The image.', use_column_width=True)
        st.write("")
        st.write("Classifying...")
        label = load_models(image)


        if label==0:
            st.write("It is a CAT")
        if label==1:
            st.write("It is a Dog")

if __name__ == "__main__":
    st.header("CAT AND DOG PREDICTION")
    st.write("Upload an image")


    upload_images()

```

## Results:

 model.fit(trainingdata,epochs=10)

```
Epoch 1/10
782/782 [=====] - 100s 128ms/step - loss: 0.6159 - accuracy: 0.6744
Epoch 2/10
782/782 [=====] - 99s 127ms/step - loss: 0.4912 - accuracy: 0.7582
Epoch 3/10
782/782 [=====] - 101s 129ms/step - loss: 0.4274 - accuracy: 0.7996
Epoch 4/10
782/782 [=====] - 100s 127ms/step - loss: 0.3772 - accuracy: 0.8295
Epoch 5/10
782/782 [=====] - 100s 128ms/step - loss: 0.3511 - accuracy: 0.8447
Epoch 6/10
782/782 [=====] - 100s 128ms/step - loss: 0.3204 - accuracy: 0.8598
Epoch 7/10
782/782 [=====] - 100s 128ms/step - loss: 0.3038 - accuracy: 0.8664
Epoch 8/10
782/782 [=====] - 99s 127ms/step - loss: 0.2918 - accuracy: 0.8721
Epoch 9/10
782/782 [=====] - 99s 127ms/step - loss: 0.2779 - accuracy: 0.8790
Epoch 10/10
782/782 [=====] - 99s 127ms/step - loss: 0.2671 - accuracy: 0.8856
<tensorflow.python.keras.callbacks.History at 0x7f3aaa308710>
```

 model.fit(trainingdata,validation\_data=validationdata,epochs=10)

```
Epoch 1/10
704/704 [=====] - 93s 131ms/step - loss: 0.6173 - accuracy: 0.6724 - val_loss: 0.5588 - val_accuracy: 0.7252
Epoch 2/10
704/704 [=====] - 92s 130ms/step - loss: 0.4907 - accuracy: 0.7578 - val_loss: 0.5131 - val_accuracy: 0.7348
Epoch 3/10
704/704 [=====] - 91s 129ms/step - loss: 0.4296 - accuracy: 0.7980 - val_loss: 0.3990 - val_accuracy: 0.8232
Epoch 4/10
704/704 [=====] - 91s 129ms/step - loss: 0.3847 - accuracy: 0.8264 - val_loss: 0.3429 - val_accuracy: 0.8432
Epoch 5/10
704/704 [=====] - 90s 128ms/step - loss: 0.3542 - accuracy: 0.8419 - val_loss: 0.3147 - val_accuracy: 0.8576
Epoch 6/10
704/704 [=====] - 89s 127ms/step - loss: 0.3309 - accuracy: 0.8520 - val_loss: 0.3279 - val_accuracy: 0.8532
Epoch 7/10
704/704 [=====] - 90s 127ms/step - loss: 0.3190 - accuracy: 0.8610 - val_loss: 0.3466 - val_accuracy: 0.8492
Epoch 8/10
704/704 [=====] - 90s 128ms/step - loss: 0.3018 - accuracy: 0.8662 - val_loss: 0.3381 - val_accuracy: 0.8596
Epoch 9/10
704/704 [=====] - 93s 131ms/step - loss: 0.2969 - accuracy: 0.8715 - val_loss: 0.3218 - val_accuracy: 0.8596
Epoch 10/10
704/704 [=====] - 92s 131ms/step - loss: 0.2792 - accuracy: 0.8791 - val_loss: 0.2868 - val_accuracy: 0.8824
<tensorflow.python.keras.callbacks.History at 0x7f89db6da6a0>
```

## GUI



## CAT AND DOG PREDICTION

Upload an image

Choose an Image ...

\_111434468\_gettyimages-1143489763.jpg

[browse files](#)



Uploaded The image.

Classifying...

It is a CAT

### Conclusion:

Thus, we have understood how and where CNN is used and how it is programmed in TensorFlow.

We have also been able to polish GUI creation skills even further.