Types of Pooling:

There are two types of padding: Same padding and Valid Padding

After every conv layer pooling should be added

Aim to apply padding and stride – To carry all information + feature map size reduce

Note: Filter greater – means data output will be more precise. But over-fitting problem appears. So standard filter size is 3x3

Formula for Output Shape Calculation: m-n+1 (here m = image size, n = filter size)

Formula for Parameter Calculation:

Keras-padding-demo.ipynb

Import libraries: tensor flow, keras

Split in train test

Create CNN model: model – sequential

- For ANN add dense layer
- For CNN add conv layer

There are three conv layers applied on 28x28 image (how many layers added depend on architecture according to need)

- Applied 3x3 filter (basic filter size, used in most cases)
- Padding type: Valid (There are two types of padding same and valid)
- Add activation function

Then next layer – to flatten data

Output layer – In dataset =10units + Then add activation function

- For Multiclass classification = activation function softmax
- For binary classification = activation function sigmoid

Model Summary:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
conv2d_3 (Conv2D)	(None, 24, 24, 32)	9,248
conv2d_4 (Conv2D)	(None, 22, 22, 32)	9,248
flatten_1 (Flatten)	(None, 15488)	0
dense_2 (Dense)	(None, 128)	1,982,592
dense_3 (Dense)	(None, 10)	1,290

Total params: 2,002,698 (7.64 MB)
Trainable params: 2,002,698 (7.64 MB)
Non-trainable params: 0 (0.00 B)

Trainable Parameters: *2,002, 698*

Formula use: m-n+1 (Image size m=28x28, Filter size n=3x3, Hidden units=32)

For Output Shape Column

- 1st layer: 28-3+1= 26 (so first layer is 26x26 from 32)
- 2nd layer: 26-3+1= 24 (so second layer is 24x24 from 32)
- 3rd layer: 24-3+1= 22 (so third layer is 22x22 from 32)

After flatten (convert image into vector means denser)

- 22 x 22 x 32 = 15488 (here 22x22 image size and 32 is hidden layer)

Then 15488 hidden units connects with dense layer where neurons = 128

- 15488 x 128 = 1982464
- Add bias: 1982464 + 128 = 1982592

At last output units = 10units

Note: If feature matrix is greater means it requires larger storage, larger consumption or computational power. So:

If we apply stride + padding: same

Feature maps become half (which reduce time consumption and computational power)

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 14, 14, 32)	320
conv2d_6 (Conv2D)	(None, 7, 7, 32)	9,248
conv2d_7 (Conv2D)	(None, 4, 4, 32)	9,248
flatten_2 (Flatten)	(None, 512)	0
dense_4 (Dense)	(None, 128)	65,664
dense_5 (Dense)	(None, 10)	1,290

Total params: 85,770 (335.04 KB)
Trainable params: 85,770 (335.04 KB)
Non-trainable params: 0 (0.00 B)

Trainable Parameters: *85, 770 (parameters reduced)*

Image size 28x28

- Half of 28x28 = 14x14
- Half of 14x14 = 7x7
- Half of 7x7 = 3.5x3.5 (round of it = 4x4)

Note: If answer came in float – round off values and always go above to avoid losing information

If we apply pooling + stride + padding: valid

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_9 (Conv2D)	(None, 11, 11, 32)	9,248
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten_3 (Flatten)	(None, 800)	0
dense_6 (Dense)	(None, 128)	102,528
dense_7 (Dense)	(None, 10)	1,290

Total params: 113,386 (442.91 KB)
Trainable params: 113,386 (442.91 KB)
Non-trainable params: 0 (0.00 B)

Trainable Parameters: *113,386*

Formula use: m-n+1 (Image size m=28x28, Filter size n=3x3, Hidden units=32)

For Output Shape Column

- 1^{st} layer: 28-3+1= 26 (so first layer is 26x26 from 32)
- Then after pooling – half of 26 =13

- 2^{nd} layer: 13-3+1= 11 (so second layer is 24x24 from 32)
- Then after pooling – half of 11 = 4.5 rounding off = 5

After flatten (convert image into vector means denser)

- 5 x 5 x 32 = 800 (here 5x5 image size and 32 is hidden layer)

In dense layer 128 neurons

At last output units = 10units

If we apply pooling + stride + padding: same

Model: "sequential_4"

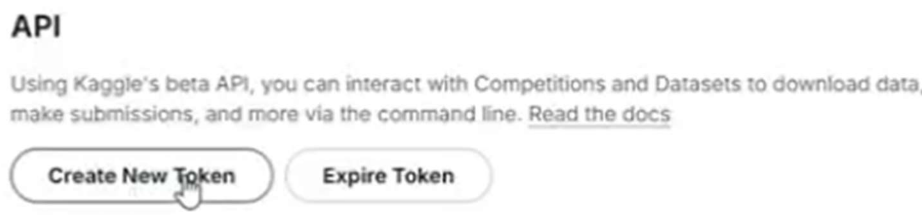
Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_11 (Conv2D)	(None, 14, 14, 32)	9,248
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 32)	0
flatten_4 (Flatten)	(None, 1568)	0
dense_8 (Dense)	(None, 128)	200,832
dense_9 (Dense)	(None, 10)	1,290

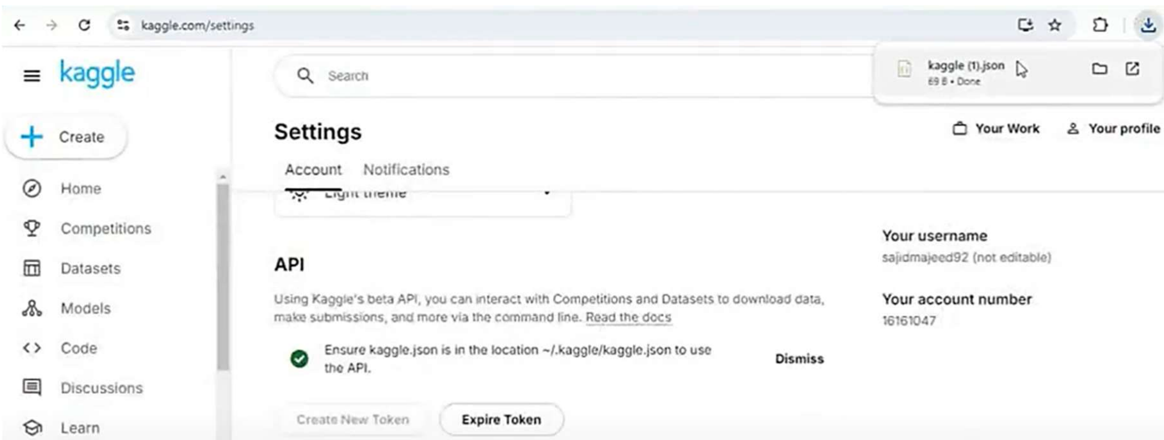
Total params: 211,690 (826.91 KB)
Trainable params: 211,690 (826.91 KB)
Non-trainable params: 0 (0.00 B)

Note: It is considering the whole system that is why Trainable parameters does not reduced.

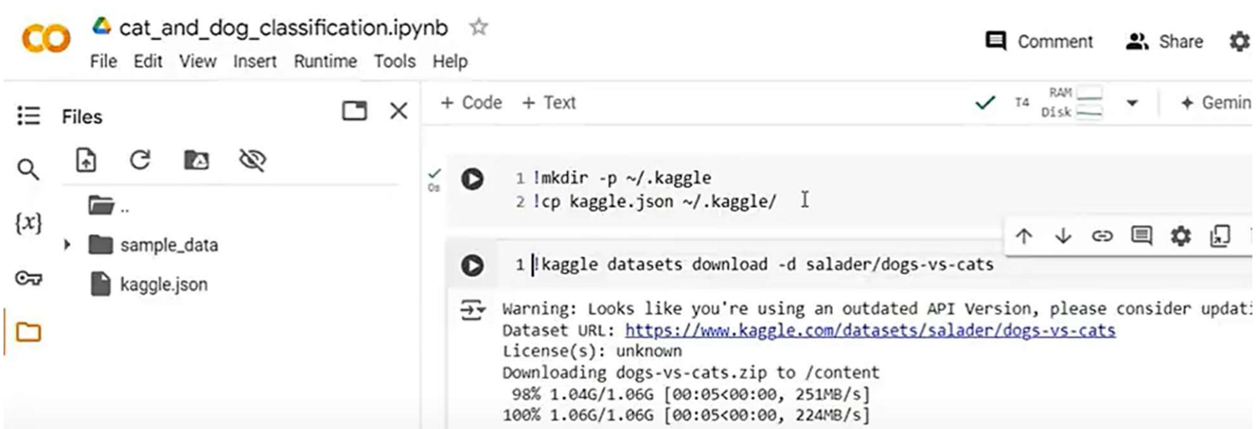
Cat VS Dog Classification

Go to kaggle: Create account – Go to Settings – API (Create new token) – File automatically download

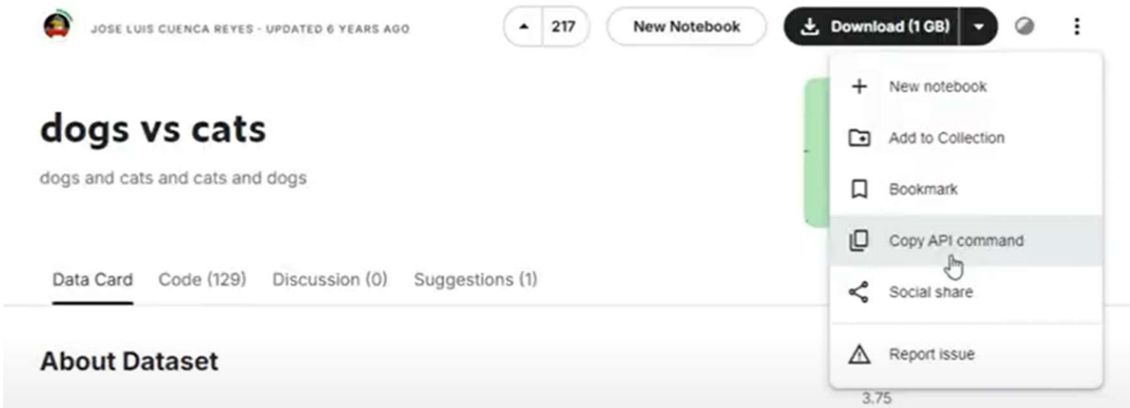




Upload this kaggle file in collab

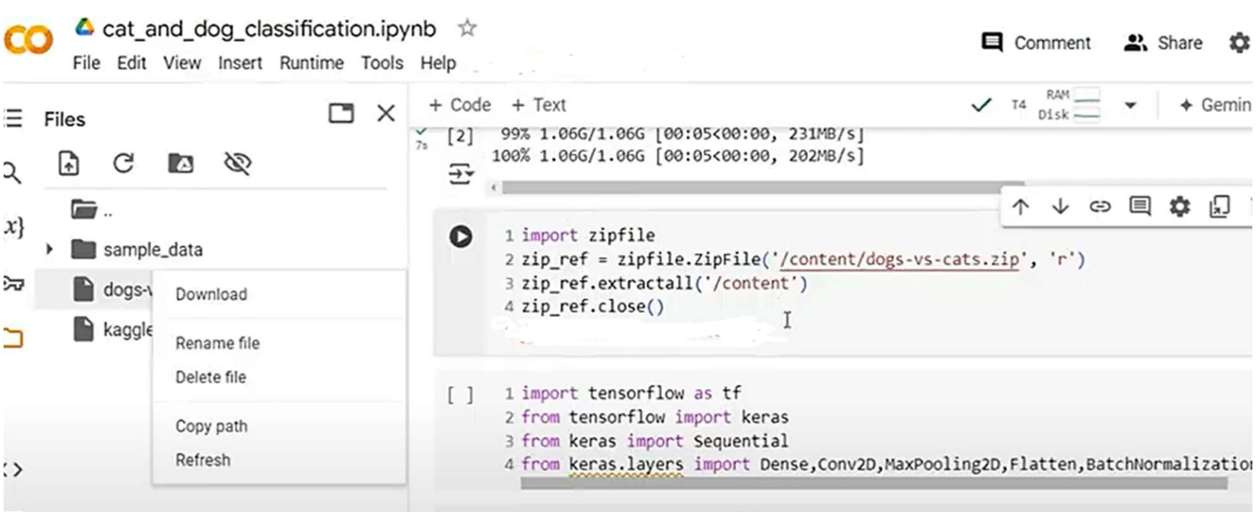


- Go to the website and copy API command: https://www.kaggle.com/datasets/salader/dogs-vs-cats
- Then with exclamation sign paste API command in collab
- Data will be downloaded in zip form



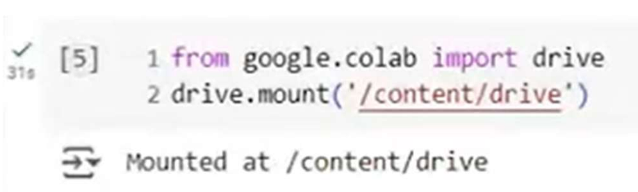
Unzip File:

- Copy path and paste + parameter r to read file
- Path where data need to add
- Closing syntax



Then again same steps for creating model

- Import libraries
- Drive mount used to link google drive with collab



Keras generator: https://keras.io/api/data_loading/image/

Kears utility: https://keras.io/api/utils/

Then, add generator code in model

Label inferred from directory and then write it in form of integer #this will done by keras generator

Note:

- In generator validation carry test data
- Here folder are already made by kaggle, but if it's not the case then we need to label it by using loop

```
# generators
train_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)

validation_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/test',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)
```

Pre-processing:

Normalize train and test images using **map function**

```
#Normalize
def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)
```

Create CNN Model + Add pooling layer + Flatten layer + Dense layer

Note: In conv layer hidden layer increase: 32,64,128 while in dense layer hidden layer decrease: 128, 64, 32

If we don't use Batch Normalization and Dropout – Data will overfit

Model Summary

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 254, 254, 32)      896

 batch_normalization_3 (Bat  (None, 254, 254, 32)      128
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 125, 125, 64)      18496

 batch_normalization_4 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 60, 60, 128)       73856

 batch_normalization_5 (Bat  (None, 60, 60, 128)       512
 chNormalization)

 max_pooling2d_5 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten_1 (Flatten)         (None, 115200)            0

 dense_3 (Dense)             (None, 128)               14745728

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

Total params: 14848193 (56.64 MB)
Trainable params: 14847745 (56.64 MB)
Non-trainable params: 448 (1.75 KB)
```

Trainable Parameters: *14847745*

Formula use: m-n+1 (Image size m=256x256, Filter size n=3x3, Hidden units=32)

For Output Shape Column

- 1^{st} layer: 256-3+1= 254 (so first layer is 254x254 from 32)
- Then after pooling – half of 254 =127

- 2^{nd} layer: 127-3+1= 125 (so second layer is 125x125 from 64)
- Then after pooling – half of 125 = 62.5 rounding off = 62

- 3^{rd} layer: 62-3+1= 60 (so second layer is 60x60 from 128)
- Then after pooling – half of 60 = 30

After flatten (convert image into vector means denser)

- 60 x 60 x 128= 115200 (here 60x60 image size and 128 is hidden layer)

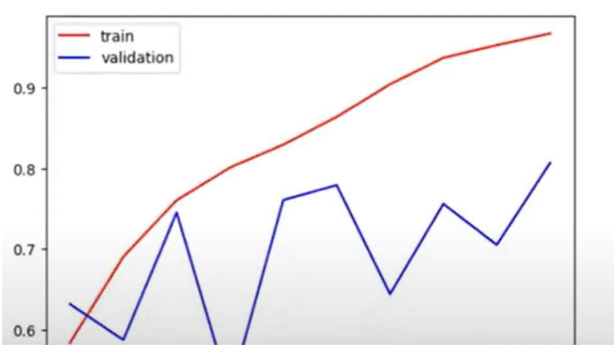
In dense layer 128 neurons

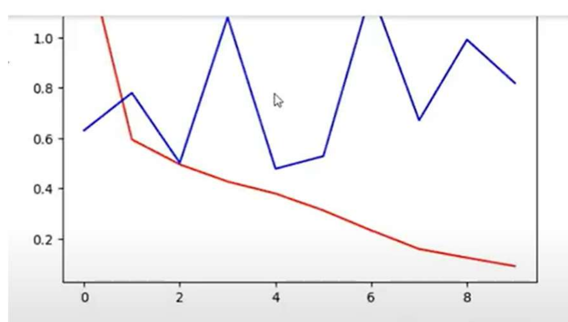
Check Losses

```
1 model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
1 history = model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

Plots without applying Batch Normalization and Dropout





Plot shows training model accuracy = 96% while testing model has accuracy = 76% means loss appear that means model overfit.

Ways to reduce overfitting

- Add more data
- Data Augmentation -> next lecture
- L1/L2 Regularizer
- Dropout
- Batch Norm
- Reduce complexity

Add More Data: Increasing the amount of training data helps the model learn more diverse patterns and generalize better to unseen data.

Data Augmentation: By applying transformations like rotation, cropping, or flipping, you create variations of your existing data, which helps the model generalize better.

L1/L2 Regularizer: Regularization techniques add penalties to the loss function for large weights (L1 and L2 norms), discouraging the model from fitting the noise in the training data.

Dropout: Dropout randomly drops a subset of neurons during training, which prevents the network from relying too heavily on any single neuron and encourages redundancy in the learned features.

Batch Norm: Normalizes the inputs to each layer, which helps stabilize and speed up training, and can act as a regularizer.

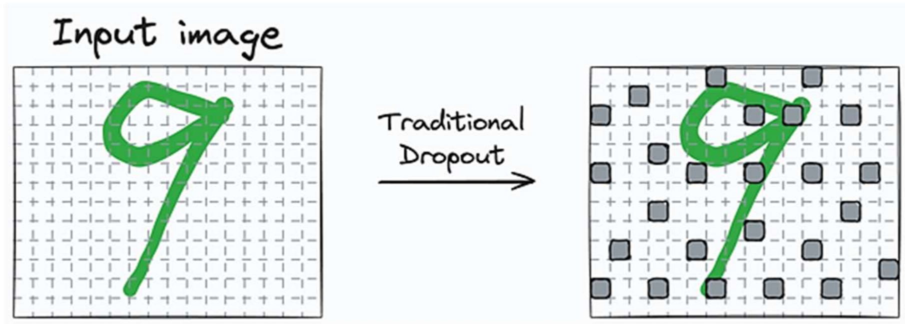
Reduce Complexity: Simplifying the model architecture (e.g., reducing the number of layers or units) can help prevent the model from memorizing the training data and encourage better generalization.

Dropout:

Dropout is a regularization technique used during the training of neural networks. The key idea is to randomly "drop out" or ignore a subset of neurons (along with their connections) during each training iteration.

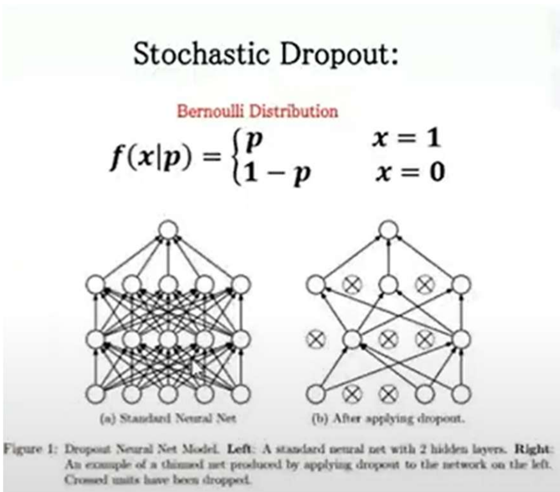
This means that each neuron has a certain probability of being excluded from the network during a particular forward and backward pass.

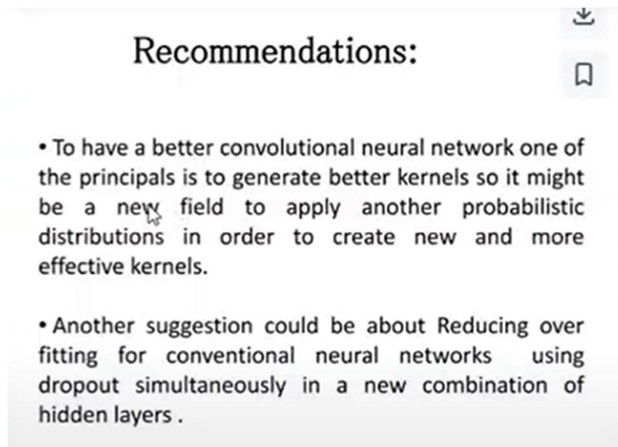
Note: Dropout layer will be added after flatten layer



Definition: It is a regularization method that stochastically sets to zero the activations of hidden units for each training case at training time.

Means neurons will participate but does not do anything, so dropout will drop them.





Batch Normalization:

Batch Normalization involves normalizing the inputs of each layer in a mini-batch. This normalization is done before applying the activation function and helps maintain stable distributions of activations throughout training.

Note: It is add after conv layer