# [**Convolutional Neural Networks in TensorFlow**](https://www.coursera.org/learn/convolutional-neural-networks-tensorflow/home/welcome)

**Week 1 - Exploring a larger dataset**

This week they basically made a review over the last course, shared some codes and trained a simple convolutional neural network for classifying images of cats and dogs.

Below there is a snippet for generating the data as input for the model training.

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.

train\_datagen = ImageDataGenerator( rescale = 1.0/255. )

test\_datagen = ImageDataGenerator( rescale = 1.0/255. )

# --------------------

# Flow training images in batches of 20 using train\_datagen generator

# --------------------

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

batch\_size=20,

class\_mode='binary',

target\_size=(150, 150))

# --------------------

# Flow validation images in batches of 20 using test\_datagen generator

# --------------------

validation\_generator = test\_datagen.flow\_from\_directory(validation\_dir,

batch\_size=20,

class\_mode = 'binary',

target\_size = (150, 150))

**Week 2 - Augmentation: A technique to avoid overfitting**

This week they basically explored the need for data augmentation, and how this can affect the results on the training and validation. Thus, avoiding overfitting in the training set.

Below there is a snippet for augmenting the data as input for the model training.

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest')

validation\_datagen = ImageDataGenerator(rescale=1/255)

**Week 3 - Transfer learning**

This week they talked about how you can take an existing model, freeze many of its layers to prevent them from being retrained, and effectively 'remember' the convolutions it was trained on.

Then you can add your own DNN underneath this so that you could retrain on your images using the convolutions from the other model.

It also explained about regularization using **dropouts** to make your network more efficient in preventing over-specialization and this overfitting.

The snippet for reading a trained model is shown below.

from tensorflow.keras import layers

from tensorflow.keras import Model

!wget --no-check-certificate \

https://storage.googleapis.com/mledu-datasets/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5 \

-O /tmp/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

from tensorflow.keras.applications.inception\_v3 import InceptionV3

local\_weights\_file = '/tmp/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'

pre\_trained\_model = InceptionV3(input\_shape = (150, 150, 3),

include\_top = False,

weights = None)

pre\_trained\_model.load\_weights(local\_weights\_file)

for layer in pre\_trained\_model.layers:

layer.trainable = False

# pre\_trained\_model.summary()

last\_layer = pre\_trained\_model.get\_layer('mixed7')

print('last layer output shape: ', last\_layer.output\_shape)

last\_output = last\_layer.output

The snippet for adding extra layers and a dropout is shown below.

from tensorflow.keras.optimizers import RMSprop

# Flatten the output layer to 1 dimension

x = layers.Flatten()(last\_output)

# Add a fully connected layer with 1,024 hidden units and ReLU activation

x = layers.Dense(1024, activation='relu')(x)

# Add a dropout rate of 0.2

x = layers.Dropout(0.2)(x)

# Add a final sigmoid layer for classification

x = layers.Dense (1, activation='sigmoid')(x)

model = Model( pre\_trained\_model.input, x)

model.compile(optimizer = RMSprop(lr=0.0001),

loss = 'binary\_crossentropy',

metrics = ['accuracy'])

**Week 4 - Multiclass Classifications**

This week they covered the problems with multiple classes, and below you have

import tensorflow as tf

import keras\_preprocessing

from keras\_preprocessing import image

from keras\_preprocessing.image import ImageDataGenerator

TRAINING\_DIR = "/tmp/rps/"

training\_datagen = ImageDataGenerator(

rescale = 1./255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest')

VALIDATION\_DIR = "/tmp/rps-test-set/"

validation\_datagen = ImageDataGenerator(rescale = 1./255)

train\_generator = training\_datagen.flow\_from\_directory(

TRAINING\_DIR,

target\_size=(150,150),

class\_mode='categorical',

batch\_size=126

)

validation\_generator = validation\_datagen.flow\_from\_directory(

VALIDATION\_DIR,

target\_size=(150,150),

class\_mode='categorical',

batch\_size=126

)

model = tf.keras.models.Sequential([

# Note the input shape is the desired size of the image 150x150 with 3 bytes color

# This is the first convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu', input\_shape=(150, 150, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

# The second convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The third convolution

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The fourth convolution

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# Flatten the results to feed into a DNN

tf.keras.layers.Flatten(),

tf.keras.layers.Dropout(0.5),

# 512 neuron hidden layer

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dense(3, activation='softmax')

])

model.summary()

model.compile(loss = 'categorical\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

history = model.fit(train\_generator, epochs=25, steps\_per\_epoch=20, validation\_data = validation\_generator, verbose = 1, validation\_steps=3)

model.save("rps.h5")