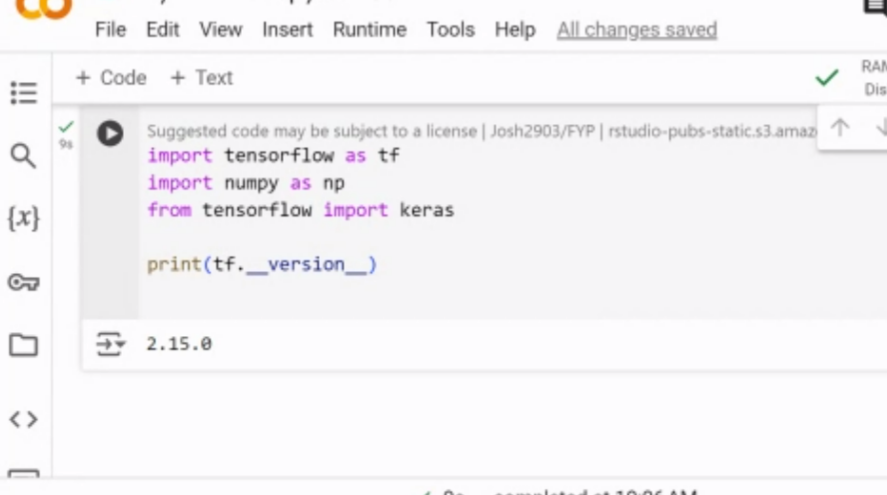
Day 1

Tensor Flow

Keras

Colab





My file: 





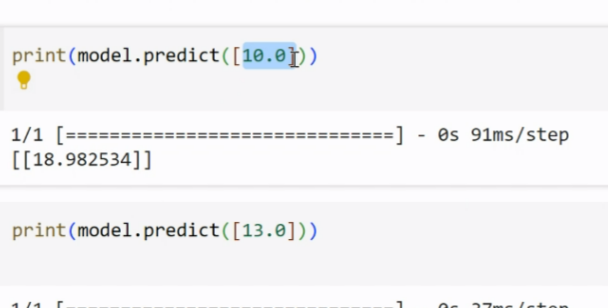
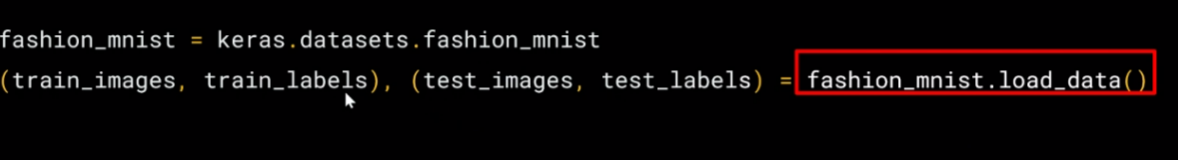


Image classification





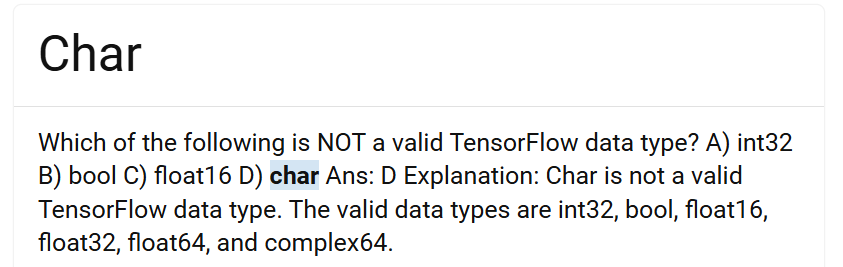
import numpy as np  
import matplotlib.pyplot as plt

index = 0  
np.set\_printoptions(linewidth=320)  
print(f'LABEL:{training\_labels[index]}')  
print(f'\nIMAGE PIXEL ARRAY:\n{training\_images[index]}')  
plt.imshow(training\_images[index])  
training\_images = training\_images / 255.0  
test\_images = test\_images / 255.0

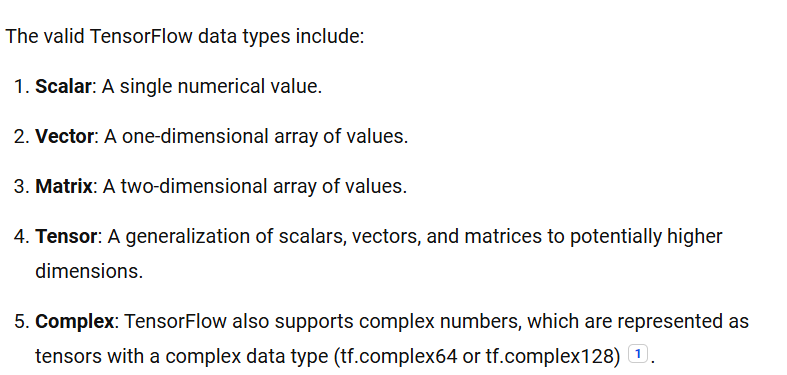
import tensorflow as tf  
print(tf.**version**)   
fmnist= tf.keras.datasets.fashion\_mnist  
(training\_images, training\_labels), (test\_images, test\_labels) = fmnist.load\_data()  
import numpy as np  
import matplotlib.pyplot as plt  
index = 0  
np.set\_printoptions(linewidth=320)  
print(f'LABEL:{training\_labels[index]}')  
print(f'\nIMAGE PIXEL ARRAY:\n{training\_images[index]}')

plt.imshow(training\_images[index])  
training\_images = training\_images / 255.0  
test\_images = test\_images / 255.0

Second Poll

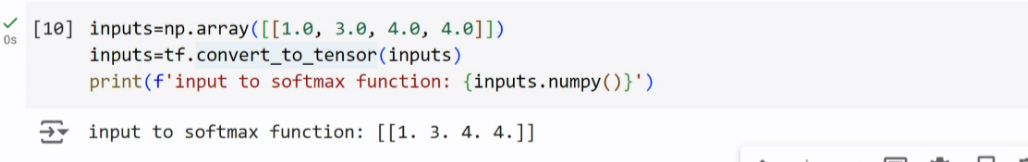


[TensorFlow MCQs and Answers With Explanation (freshersnow.com)](https://www.freshersnow.com/tensorflow-mcqs-and-answers-with-explanation/#:~:text=Which%20of%20the%20following%20is%20NOT%20a%20valid,are%20int32%2C%20bool%2C%20float16%2C%20float32%2C%20float64%2C%20and%20complex64.)



**DAY 2**

model=tf.keras.models.Sequential([tf.keras.layers.flatten(),  
tf.keras.layers.Dense(128,activation=tf.nn.relu(),)]  
tf.keras.layers.Dense(10,activation=tf.nn.softmax())])

****

inputs=np.array([1.0,3.0,4.0,4.0])  
inputs=tf.convert\_to\_tensor(inputs)  
print(f'input to softmax:{inputs.numpy()}')

outputs=tf.keras.activations.softmax(inputs)  
print(f'output of softmax function: {outputs.numpy()}')

model.compile(optimizer=tf.optimizers.Adam(),  
              loss='sparse\_categorical\_crossentropy',  
              metrics=['accuracy'])

model.fit(training\_images, training\_labels, epochs=5)

**Evaluating model on unseen DATA**

#Load the Fashion MNIST dataset  
fmnist = tf.keras.datasets.fashion\_mnist

# Load the training and test split of the Fashion MNIST dataset

(training\_images, training\_labels), (test\_images, test\_labels) = fmnist.load\_data()  
##print a training image (both as an image and a numpy array), and a training label to see:  
  
import numpy as np  
import matplotlib.pyplot as plt

# You can put between 0 to 59999 here

index = 0

# Set number of characters per row when printing

np.set\_printoptions(linewidth=320)

# Print the label and image

print(f'LABEL: {training\_labels[index]}')  
print(f'\nIMAGE PIXEL ARRAY:\n {training\_images[index]}')

# Visualize the image

plt.imshow(training\_images[index])

# Normalize the pixel values of the train and test images

training\_images = training\_images / 255.0  
test\_images = test\_images / 255.0

# Build the classification model

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),  
tf.keras.layers.Dense(128, activation=tf.nn.relu),  
tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

# Declare sample inputs and convert to a tensor

inputs = np.array([[1.0, 3.0, 4.0, 2.0]])  
inputs = tf.convert\_to\_tensor(inputs)  
print(f'input to softmax function: {inputs.numpy()}')

# Feed the inputs to a softmax activation function

outputs = tf.keras.activations.softmax(inputs)  
print(f'output of softmax function: {outputs.numpy()}')

# Get the sum of all values after the softmax

sum = tf.reduce\_sum(outputs)  
print(f'sum of outputs: {sum}')

# Get the index with highest value

prediction = np.argmax(outputs)  
print(f'class with highest probability: {prediction}')  
  
##compiling it with an optimizer and loss function  
model.compile(optimizer = tf.optimizers.Adam(),  
loss = 'sparse\_categorical\_crossentropy',  
metrics=['accuracy'])  
  
## train it by calling [model.fit](https://model.fit)()  
model.fit(training\_images, training\_labels, epochs=5).

# Evaluate the model on unseen data

model.evaluate(test\_images, test\_labels)

**QUES:** Let's now look at the layers in your model. Experiment with different values for the dense layer with 512 neurons. What different results do you get for loss, training time etc? Why do you think that's the case?

**Call back:**

import tensorflow as tf

# Instantiate the dataset API

fmnist = tf.keras.datasets.fashion\_mnist

# Load the dataset

(x\_train, y\_train),(x\_test, y\_test) = fmnist.load\_data()

# Normalize the pixel values

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Creating a Callback class

class myCallback(tf.keras.callbacks.Callback):  
def on\_epoch\_end(self, epoch, logs={}):  
'''  
Halts the training after reaching 60 percent accuracy  
  
Args:  
epoch (integer) - index of epoch (required but unused in the function definition below)  
logs (dict) - metric results from the training epoch  
'''  
  
# Check accuracy  
if(logs.get('loss') < 0.4):  
  
# Stop if threshold is met  
print("\nLoss is lower than 0.4 so cancelling training!")  
self.model.stop\_training = True

# Instantiate class

callbacks = myCallback()

# Define and compile the model

# Define the model

model = tf.keras.models.Sequential([  
tf.keras.layers.Flatten(input\_shape=(28, 28)),  
tf.keras.layers.Dense(512, activation=tf.nn.relu),  
tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])

# Compile the model

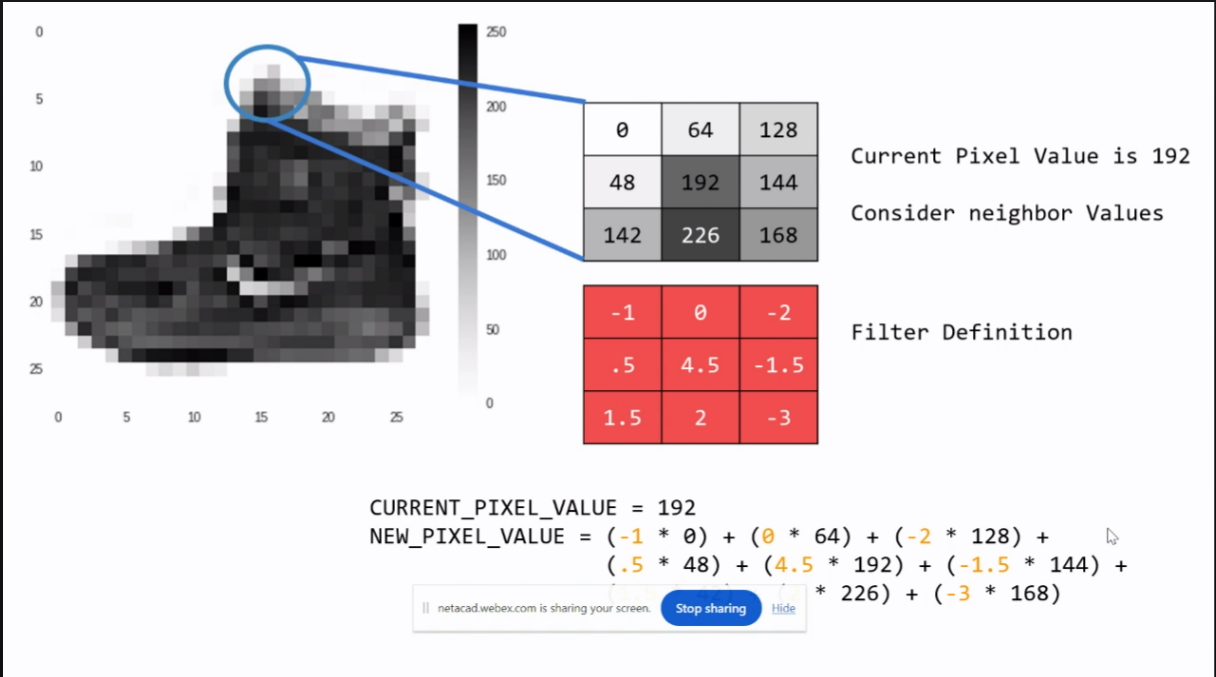
model.compile(optimizer=tf.optimizers.Adam(),  
loss='sparse\_categorical\_crossentropy',  
metrics=['accuracy'])  
#Train the model

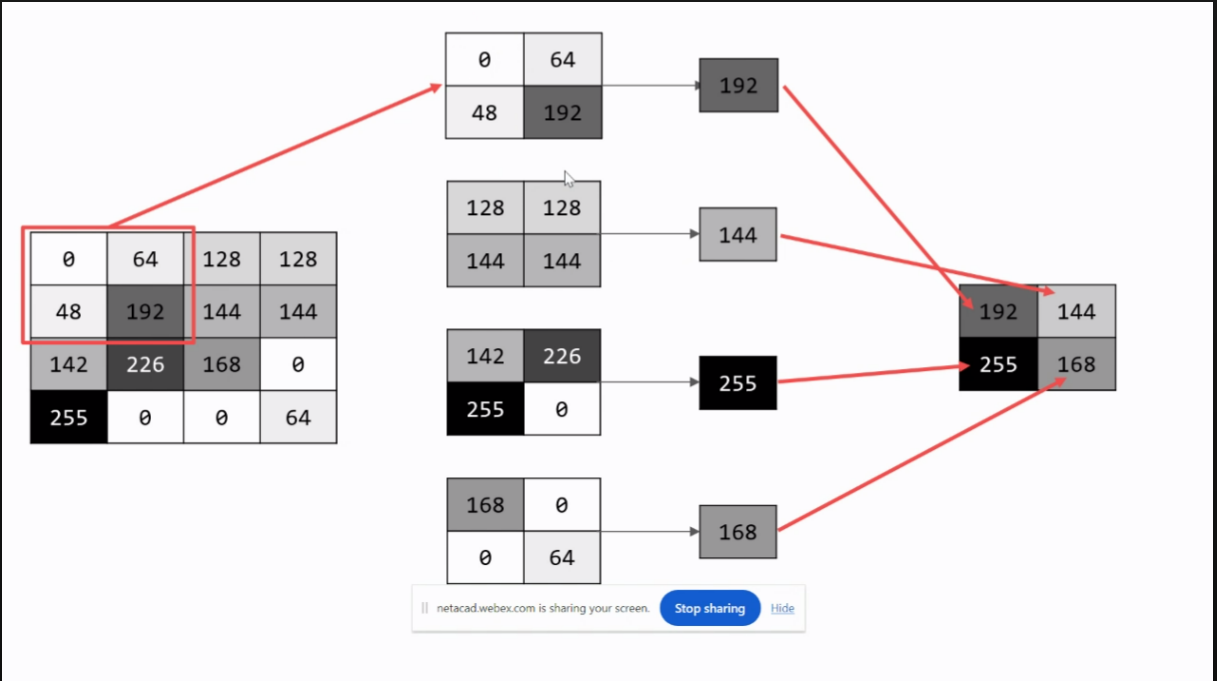
# Train the model with a callback

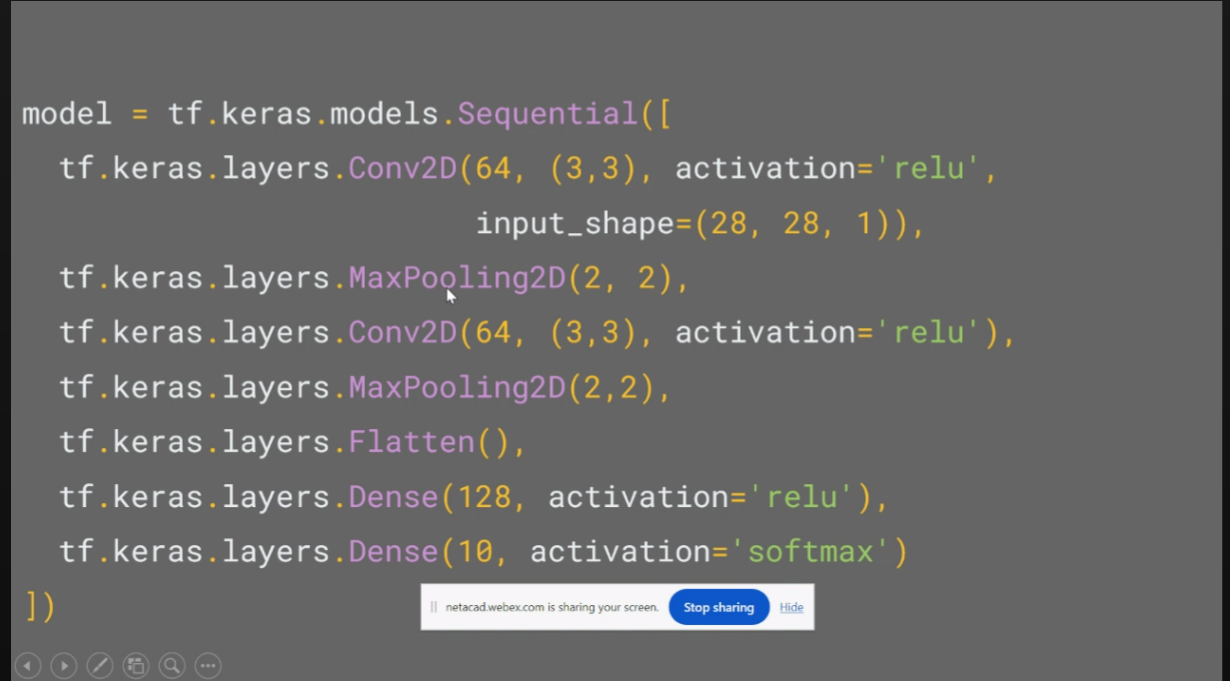
model.fit(x\_train, y\_train, epochs=10, callbacks=[callbacks])

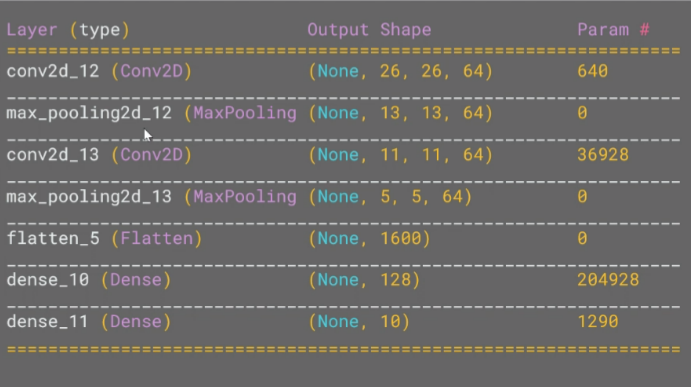
Optional Challenge: Modify the code to make the training stop when the accuracy metric exceeds 60%.

DAY 3









Adil yousuf 27-06-2024 10:04 •

import tensorflow as tf

# Load the Fashion MNIST dataset

fmnist = tf.keras.datasets.fashion\_mnist  
(training\_images, training\_labels), (test\_images, test\_labels) = fmnist.load\_data()

# Normalize the pixel values

training\_images = training\_images / 255.0  
test\_images = test\_images / 255.0

# Define the model

model = tf.keras.models.Sequential([  
tf.keras.layers.Flatten(),  
tf.keras.layers.Dense(128, activation=tf.nn.relu),  
tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])

# Setup training parameters

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

print(f'\nMODEL TRAINING:')  
model.fit(training\_images, training\_labels, epochs=5)

# Evaluate on the test set

print(f'\nMODEL EVALUATION:')  
test\_loss = model.evaluate(test\_images, test\_labels)

Adil yousuf 27-06-2024 10:08 •

# Define the model

model = tf.keras.models.Sequential([  
  
# Add convolutions and max pooling  
tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(28, 28, 1)),  
tf.keras.layers.MaxPooling2D(2, 2),  
tf.keras.layers.Conv2D(32, (3,3), activation='relu'),  
tf.keras.layers.MaxPooling2D(2,2),  
  
# Add the same layers as before  
tf.keras.layers.Flatten(),  
tf.keras.layers.Dense(128, activation='relu'),  
tf.keras.layers.Dense(10, activation='softmax')  
])

# Print the model summary

model.summary()

# Use same settings

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

print(f'\nMODEL TRAINING:')  
model.fit(training\_images, training\_labels, epochs=5)

# Evaluate on the test set

print(f'\nMODEL EVALUATION:')  
test\_loss = model.evaluate(test\_images, test\_labels)  
#Visualizing the Convolutions and Pooling  
print(test\_labels[:100])  
import matplotlib.pyplot as plt  
from tensorflow.keras import models  
  
f, axarr = plt.subplots(3,4)  
  
FIRST\_IMAGE=0  
SECOND\_IMAGE=23  
THIRD\_IMAGE=28  
CONVOLUTION\_NUMBER = 1  
  
layer\_outputs = [layer.output for layer in model.layers]  
activation\_model = tf.keras.models.Model(inputs = model.input, outputs = layer\_outputs)  
  
for x in range(0,4):  
f1 = activation\_model.predict(test\_images[FIRST\_IMAGE].reshape(1, 28, 28, 1))[x]  
axarr[0,x].imshow(f1[0, : , :, CONVOLUTION\_NUMBER], cmap='inferno')  
axarr[0,x].grid(False)  
  
f2 = activation\_model.predict(test\_images[SECOND\_IMAGE].reshape(1, 28, 28, 1))[x]  
axarr[1,x].imshow(f2[0, : , :, CONVOLUTION\_NUMBER], cmap='inferno')  
axarr[1,x].grid(False)  
  
f3 = activation\_model.predict(test\_images[THIRD\_IMAGE].reshape(1, 28, 28, 1))[x]  
axarr[2,x].imshow(f3[0, : , :, CONVOLUTION\_NUMBER], cmap='inferno')  
axarr[2,x].grid(False)

**Exercise:**

1. Try editing the convolutions. Change the 32s to either 16 or 64. What impact will this have on accuracy and/or training time.
2. Remove the final Convolution. What impact will this have on accuracy or training time?
3. How about adding more Convolutions? What impact do you think this will have? Experiment with it.

[MCQs | Convolutional Neural Networks | AIMCQs](https://aimcqs.com/convolutional-neural-networks)

DAY 4

IMAGE DATA GENERATION

!wget https://storage.googleapis.com/tensorflow-1-public/course2/week3/horse-or-human.zip

import zipfile

# Unzip the dataset

local\_zip = './horse-or-human.zip'  
zip\_ref = zipfile.ZipFile(local\_zip, 'r')  
zip\_ref.extractall('./horse-or-human')  
zip\_ref.close()

import os

# Directory with our training horse pictures

train\_horse\_dir = os.path.join('./horse-or-human/horses')

# Directory with our training human pictures

train\_human\_dir = os.path.join('./horse-or-human/humans')

train\_horse\_names = os.listdir(train\_horse\_dir)  
print(train\_horse\_names[:10])  
   
train\_human\_names = os.listdir(train\_human\_dir)  
print(train\_human\_names[:10])

print('total training horse images:', len(os.listdir(train\_horse\_dir)))  
print('total training human images:', len(os.listdir(train\_human\_dir)))

Adil yousuf 28-06-2024 10:16 •

%matplotlib inline  
  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg

# Parameters for our graph; we'll output images in a 4x4 configuration

nrows = 4  
ncols = 4

# Index for iterating over images

pic\_index = 0

# Now, display a batch of 8 horse and 8 human pictures. You can rerun the cell to see a fresh batch each time:

# Set up matplotlib fig, and size it to fit 4x4 pics

fig = plt.gcf()  
fig.set\_size\_inches(ncols \* 4, nrows \* 4)  
  
pic\_index += 8  
next\_horse\_pix = [os.path.join(train\_horse\_dir, fname)  
for fname in train\_horse\_names[pic\_index-8:pic\_index]]  
next\_human\_pix = [os.path.join(train\_human\_dir, fname)  
for fname in train\_human\_names[pic\_index-8:pic\_index]]  
  
for i, img\_path in enumerate(next\_horse\_pix+next\_human\_pix):  
# Set up subplot; subplot indices start at 1  
sp = plt.subplot(nrows, ncols, i + 1)  
sp.axis('Off') # Don't show axes (or gridlines)  
  
img = mpimg.imread(img\_path)  
plt.imshow(img)  
  
[plt.show](https://plt.show)()

import tensorflow as tf

model = tf.keras.models.Sequential([  
# Note the input shape is the desired size of the image 300x300 with 3 bytes color  
    # This is the first convolution  
tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(300, 300, 3)),  
tf.keras.layers.MaxPooling2D(2, 2),  
    # The second convolution  
tf.keras.layers.Conv2D(32, (3,3), activation='relu'),  
tf.keras.layers.MaxPooling2D(2,2),  
    # The third convolution  
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  
tf.keras.layers.MaxPooling2D(2,2),  
    # The fourth convolution  
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  
tf.keras.layers.MaxPooling2D(2,2),  
    # The fifth convolution  
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  
tf.keras.layers.MaxPooling2D(2,2),  
    # Flatten the results to feed into a DNN  
tf.keras.layers.Flatten(),  
# 512 neuron hidden layer  
tf.keras.layers.Dense(512, activation='relu'),  
# Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') and 1 for the other ('humans')  
tf.keras.layers.Dense(1, activation='sigmoid')  
])

model.summary()

rom tensorflow.keras.optimizers import RMSprop  
   
model.compile(loss='binary\_crossentropy',  
optimizer=RMSprop(learning\_rate=0.001),  
              metrics=['accuracy'])

Adil yousuf 28-06-2024 10:31 •

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255

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

# Flow training images in batches of 128 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(  
'./horse-or-human/', # This is the source directory for training images  
target\_size=(300, 300), # All images will be resized to 300x300  
batch\_size=128,  
# Since we use binary\_crossentropy loss, we need binary labels  
class\_mode='binary')

import numpy as np  
from google.colab import files  
from keras.preprocessing import image  
   
uploaded = files.upload()  
   
for fn in uploaded.keys():  
  
  # predicting images  
path = '/content/' + fn  
img = image.load\_img(path, target\_size=(300, 300))  
x = image.img\_to\_array(img)  
x /= 255  
x = np.expand\_dims(x, axis=0)  
   
images = np.vstack([x])  
classes = model.predict(images, batch\_size=10)  
print(classes[0])  
      
if classes[0]>0.5:  
    print(fn + " is a human")  
else:  
    print(fn + " is a horse")

UPLOAD ANY IMAGE AND CHECK FOR ACCURACY

import numpy as np  
import random  
from tensorflow.keras.preprocessing.image import img\_to\_array, load\_img

# Define a new Model that will take an image as input, and will output

# intermediate representations for all layers in the previous model after

# the first.

successive\_outputs = [layer.output for layer in model.layers[1:]]  
visualization\_model = tf.keras.models.Model(inputs = model.input, outputs = successive\_outputs)

28-06-2024 11:29 •

# Prepare a random input image from the training set.

horse\_img\_files = [os.path.join(train\_horse\_dir, f) for f in train\_horse\_names]  
human\_img\_files = [os.path.join(train\_human\_dir, f) for f in train\_human\_names]  
img\_path = random.choice(horse\_img\_files + human\_img\_files)  
  
img = load\_img(img\_path, target\_size=(300, 300)) # this is a PIL image  
x = img\_to\_array(img) # Numpy array with shape (300, 300, 3)  
x = x.reshape((1,) + x.shape) # Numpy array with shape (1, 300, 300, 3)

# Scale by 1/255

x /= 255

# Run the image through the network, thus obtaining all

# intermediate representations for this image.

successive\_feature\_maps = visualization\_model.predict(x)

# These are the names of the layers, so you can have them as part of the plot

layer\_names = [layer.name for layer in model.layers[1:]]

**Display the representations**

for layer\_name, feature\_map in zip(layer\_names, successive\_feature\_maps):  
if len(feature\_map.shape) == 4:  
  
# Just do this for the conv / maxpool layers, not the fully-connected layers  
n\_features = feature\_map.shape[-1] # number of features in feature map  
  
# The feature map has shape (1, size, size, n\_features)  
size = feature\_map.shape[1]  
  
# Tile the images in this matrix  
display\_grid = np.zeros((size, size \* n\_features))  
for i in range(n\_features):  
x = feature\_map[0, :, :, i]  
x -= x.mean()  
x /= x.std()  
x \*= 64  
x += 128  
x = np.clip(x, 0, 255).astype('uint8')  
  
# Tile each filter into this big horizontal grid  
display\_grid[:, i \* size : (i + 1) \* size] = x  
  
# Display the grid  
scale = 20. / n\_features  
plt.figure(figsize=(scale \* n\_features, scale))  
plt.title(layer\_name)  
plt.grid(False)  
plt.imshow(display\_grid, aspect='auto', cmap='viridis')

import os, signal

os.kill(os.getpid(),signal.SIGKILL)