exp-1-ml

April 23, 2025

[2]:
```
import numpy as np
```

[3]:
```
X = np.array([[0, 0, 1, 1],
              [0, 1, 0, 1]])
W = np.array([1, 1])                  #Polar OR
theta = 1
Z = W@X
y_pred = [int(i>=theta) for i in Z]
print(y_pred)
```

```
[0, 1, 1, 1]
```

[4]:
```
X = np.array([[0, 0, 1, 1],
              [0, 1, 0, 1]])
W = np.array([1, 1])
theta = 2
Z = W@X                                         #Polar AND
y_pred = [int(i>=theta) for i in Z]
print(y_pred)
```

```
[0, 0, 0, 1]
```

[5]:
```
X = np.array([[-1, -1, 1, 1],
              [-1, 1, -1, 1]])
W = np.array([1, 1])
theta = 0                                     #Bipolar OR
Z = W@X
y_pred = [int(i>=theta)*2-1 for i in Z]
print(y_pred)
```

```
[-1, 1, 1, 1]
```

[6]:
```
X = np.array([[-1, -1, 1, 1],
              [-1, 1, -1, 1]])
W = np.array([1, 1])                   #Bipolar AND
theta = 1
Z = W@X
y_pred = [int(i>=theta)*2-1 for i in Z]
```

```python
print(y_pred)
```

```
[-1, -1, -1, 1]
```

[ ]:

```python

```

[ ]:

```python

```

mlp-backward-pass-2-layered-1

April 23, 2025

[7]:
```python
import numpy as np
np.set_printoptions(precision=4)

# Initialize weights
W_0 = np.array([[1, 0, 1],
                [-1, -1, 1]], dtype=float)  # Hidden layer weights
W_1 = np.array([[0, 1, -1]], dtype=float)   # Output layer weights

# Input and target
X = np.array([[1, -1, 1],
              [0, -1, 1]], dtype=float)     # Shape: (2, 2)
t = np.array([[0, 0, 1]], dtype=float)   # Shape: (1, 2)

# Hyperparameters
f_1 = "Lin"
f_2 = "ReLU"
lr = 1
MAX_EPOCHS = 1

# Activation Functions
def USigmoid(x, direction='F'):
    fx = 1 / (1 + np.exp(-x))
    return fx if direction == 'F' else fx * (1 - fx)

def BSigmoid(x, direction='F'):
    fx = (1 - np.exp(-x)) / (1 + np.exp(-x))
    return fx if direction == 'F' else 0.5 * (1 - fx ** 2)

def TanH(x, direction='F'):
    fx = np.tanh(x)
    return fx if direction == 'F' else 1 - fx ** 2

def ReLU(x, direction='F'):
    return np.maximum(0, x) if direction == 'F' else (x > 0).astype(float)

def Lin(x, direction='F'):
    return x if direction == 'F' else np.ones_like(x)
```

```python
# Wrapper for activation
def activation(Z, fcn="Lin", direction='F'):
    return np.array([globals()[fcn](z, direction) for z in Z]).reshape(-1, 1)

A_0 = np.vstack((np.ones((1, 1)), x.reshape(-1, 1)))

# Training loop
for ep in range(MAX_EPOCHS):
    print('\nEPOCH-', ep + 1, '=' * 80)
    for itr, (x, y) in enumerate(zip(X.T, t.T)):
        print(f'\nITER-{itr + 1} ' + '-' * 80)

        # Forward Pass: Input -> Hidden
        A_0 = x.reshape(-1, 1)

        # Conditionally add bias to A_0
        if W_0.shape[1] == A_0.shape[0] + 1:
            A_0 = np.vstack((np.ones((1, 1)), A_0))  # Add bias at the top
            print('Bias added to A_0')

        Z_1 = W_0 @ A_0
        A_1 = activation(Z_1, f_1)

        # Forward Pass: Hidden -> Output
        # Conditionally add bias to A_1
        if W_1.shape[1] == A_1.shape[0] + 1:
            A_1 = np.vstack((np.ones((1, 1)), A_1))  # Add bias at the top
            print('Bias added to A_1')

        Z_2 = W_1 @ A_1
        A_2 = activation(Z_2, f_2)

        print(f'A_0 (input):\n{A_0}')
        print(f'Z_1 (hidden pre-activation):\n{Z_1}')
        print(f'A_1 (hidden post-activation):\n{A_1}')
        print(f'Z_2 (output pre-activation):\n{Z_2}')
        print(f'A_2 (output post-activation):\n{A_2}')

       # Backward Pass: Output -> Hidden
        delta_2 = (A_2 - y.reshape(-1, 1)) * activation(Z_2, f_2, 'B')
        dW_1 = delta_2 @ A_1.T

        # Remove bias from W_1 if necessary
        if W_1.shape[1] == Z_1.shape[0] + 1:
            W_1_no_bias = W_1[:, 1:]
            print("Bias column removed from W_1 for backpropagation.")
```

```python
        else:
            W_1_no_bias = W_1

        print(W_1_no_bias)

        # Backward Pass: Hidden -> Input
        delta_1 = (W_1_no_bias.T @ delta_2) * activation(Z_1, f_1, 'B')
        dW_0 = delta_1 @ A_0.T

        # Update weights
        W_1 -= lr * dW_1
        W_0 -= lr * dW_0

        print(f'dW_1:\n{dW_1}')
        print(f'dW_0:\n{dW_0}')
        print(f'Updated W_1:\n{W_1}')
        print(f'Updated W_0:\n{W_0}')
```

```
EPOCH- 1
================================================================================

ITER-1
--------------------------------------------------------------------------------
Bias added to A_0
Bias added to A_1
A_0 (input):
[[1.]
 [1.]
 [0.]]
Z_1 (hidden pre-activation):
[[ 1.]
 [-2.]]
A_1 (hidden post-activation):
[[ 1.]
 [ 1.]
 [-2.]]
Z_2 (output pre-activation):
[[3.]]
A_2 (output post-activation):
[[3.]]
Bias column removed from W_1 for backpropagation.
[[ 1. -1.]]
dW_1:
[[ 3.  3. -6.]]
dW_0:
[[ 3.  3.  0.]
```

```
 [-3. -3.  0.]]
Updated W_1:
[[-3. -2.  5.]]
Updated W_0:
[[-2. -3.  1.]
 [ 2.  2.  1.]]

ITER-2
--------------------------------------------------------------------------------
Bias added to A_0
Bias added to A_1
A_0 (input):
[[ 1.]
 [-1.]
 [-1.]]
Z_1 (hidden pre-activation):
[[ 0.]
 [-1.]]
A_1 (hidden post-activation):
[[ 1.]
 [ 0.]
 [-1.]]
Z_2 (output pre-activation):
[[-8.]]
A_2 (output post-activation):
[[0.]]
Bias column removed from W_1 for backpropagation.
[[-2.  5.]]
dW_1:
[[0. 0. 0.]]
dW_0:
[[0. 0. 0.]
 [0. 0. 0.]]
Updated W_1:
[[-3. -2.  5.]]
Updated W_0:
[[-2. -3.  1.]
 [ 2.  2.  1.]]

ITER-3
--------------------------------------------------------------------------------
Bias added to A_0
Bias added to A_1
A_0 (input):
[[1.]
 [1.]
 [1.]]
Z_1 (hidden pre-activation):
```

```
[[-4.]
 [ 5.]]
A_1 (hidden post-activation):
[[ 1.]
 [-4.]
 [ 5.]]
Z_2 (output pre-activation):
[[30.]]
A_2 (output post-activation):
[[30.]]
Bias column removed from W_1 for backpropagation.
[[-2.  5.]]
dW_1:
[[  29. -116.  145.]]
dW_0:
[[-58. -58. -58.]
 [145. 145. 145.]]
Updated W_1:
[[ -32.  114. -140.]]
Updated W_0:
[[  56.   55.   59.]
 [-143. -143. -144.]]
```

gradient-tape-activations7

April 23, 2025

[ ]:
```
# import numpy as np

# # Initial weights
# W_0 = np.array([[1, 0, 1]], dtype=float)
# print("Initial W_0 shape:", W_0.shape)  # (1, 3)

# # Targets
# t = np.array([[1 ,-1 ,-1]], dtype=float)

# # Inputs
# X = np.array([
#     [1, 1, -1],
#     [0, 1, 1]
# ], dtype=float)
# print("X shape:", X.shape)  # (2, 3)

# # Activation function name
# f_1 = "Lin"

# # Learning rate
# lr = 0.1

# # Activation Functions
# def USigmoid(x, direction):
#     if direction == 'F':
#         return 1 / (1 + np.exp(-x))
#     else:  # derivative
#         fx = USigmoid(x, 'F')
#         return fx * (1 - fx)

# def BSigmoid(x, direction):
#     if direction == 'F':
#         return (1 - np.exp(-x)) / (1 + np.exp(-x))
#     else:
#         fx = BSigmoid(x, 'F')
#         return 0.5 * (1 - fx ** 2)
```

```
# def ReLU(x, direction):
#     if direction == 'F':
#         return np.maximum(0, x)
#     else:
#         return float(x > 0)

# def Lin(x, direction):
#     if direction == 'F':
#         return x
#     else:
#         return 1

# # Activation wrapper
# def activation(Z, fcn="Lin", direction='F'):
#     Z = np.atleast_1d(Z)
#     return np.array([globals()[fcn](z, direction) for z in Z])

# # Training loop
# MAX_EPOCH = 1

# # Pad bias term to input
# print('Pad bias at top of input')
# A_0 = np.vstack((np.ones((1, X.shape[1])), X))
# print(A_0)

# for ep in range(MAX_EPOCH):
#     print('\nEPOCH-', ep + 1, '=' * 80)
#     for itr, (x, y) in enumerate(zip(A_0.T, t.T)):
#         print('\nITER-', itr + 1, '-' * 80)

#         print(f'{y = }')

#         # Forward pass
#         print('Input -> Output')
#         Z_1 = W_0 @ x
#         print(f'{Z_1 = }')
#         A_1 = activation(Z_1, f_1)
#         print(f'{A_1 = }')

#         # Backward pass
#         print('\nOutput -> Input')
#         Error = 0.5 * (A_1 - y) ** 2
#         print(f'{Error = }')
#         dE_dW = (A_1 - y) * activation(Z_1, f_1, 'B') * x
#         print(f'{dE_dW = }')

#         # Weight update
```

```python
#         W_0 = W_0 - lr * dE_dW
#         print(f'{W_0 = }')
```

```
Initial W_0 shape: (1, 3)
X shape: (2, 3)
Pad bias at top of input
[[ 1.  1.  1.]
 [ 1.  1. -1.]
 [ 0.  1.  1.]]

EPOCH- 1
================================================================================

ITER- 1
--------------------------------------------------------------------------------
y = array([1.])
Input -> Output
Z_1 = array([1.])
A_1 = array([1.])

Output -> Input
Error = array([0.])
dE_dW = array([0., 0., 0.])
W_0 = array([[1., 0., 1.]])

ITER- 2
--------------------------------------------------------------------------------
y = array([-1.])
Input -> Output
Z_1 = array([2.])
A_1 = array([2.])

Output -> Input
Error = array([4.5])
dE_dW = array([3., 3., 3.])
W_0 = array([[ 0.7, -0.3,  0.7]])

ITER- 3
--------------------------------------------------------------------------------
y = array([-1.])
Input -> Output
Z_1 = array([1.7])
A_1 = array([1.7])

Output -> Input
Error = array([3.645])
dE_dW = array([ 2.7, -2.7,  2.7])
```

```
W_0 = array([[ 0.43, -0.03,  0.43]])
```

[4]:
```python
import numpy as np
import tensorflow as tf

# Initial weights
W_0 = tf.Variable([[1.0, 0.0, 1.0]], dtype=tf.float32)
print("Initial W_0 shape:", W_0.shape)

# Targets
t = tf.constant([[1.0, -1.0, -1.0]], dtype=tf.float32)

# Inputs
X = tf.constant([
    [1.0, 1.0, -1.0],
    [0.0, 1.0, 1.0]
], dtype=tf.float32)
print("X shape:", X.shape)

# Activation function name
f_1 = "Lin"

# Learning rate
lr = 0.1

# Activation Functions
def USigmoid(x):
    return tf.math.sigmoid(x)

def USigmoid_deriv(x):
    fx = tf.math.sigmoid(x)
    return fx * (1 - fx)

def BSigmoid(x):
    return (1 - tf.exp(-x)) / (1 + tf.exp(-x))

def BSigmoid_deriv(x):
    fx = BSigmoid(x)
    return 0.5 * (1 - tf.square(fx))

def ReLU(x):
    return tf.nn.relu(x)

def ReLU_deriv(x):
    return tf.cast(x > 0, tf.float32)

def Lin(x):
```

```
    return x

def Lin_deriv(x):
    return tf.ones_like(x)

# Activation function dictionary
activation_map = {
    "USigmoid": (USigmoid, USigmoid_deriv),
    "BSigmoid": (BSigmoid, BSigmoid_deriv),
    "ReLU": (ReLU, ReLU_deriv),
    "Lin": (Lin, Lin_deriv),
}

# Select activation function and its derivative
activation_fn, activation_deriv = activation_map[f_1]

# Add bias term to input (pad bias row at the top)
A_0 = np.vstack((np.ones((1, X.shape[1])), X))
A_0 = tf.constant(A_0, dtype=tf.float32)

# Training loop
MAX_EPOCH = 1

# Epoch loop
# Epoch loop
for ep in range(MAX_EPOCH):
    print(f"\nEPOCH-{ep + 1} " + "=" * 80)

    for itr, (x, y) in enumerate(zip(tf.transpose(A_0), tf.transpose(t))):
        print(f"\nITER-{itr + 1} " + "-" * 80)

        # Forward pass
        Z_1 = tf.matmul(W_0, tf.reshape(x, [-1, 1]))
        A_1 = activation_fn(Z_1)

        print(f"Target (y): {y.numpy()[0]:.2f}")
        print(f"Z_1 (Weighted sum): {Z_1.numpy()[0][0]:.4f}")
        print(f"A_1 (Activated output): {A_1.numpy()[0][0]:.4f}")

        # Error and gradient
        Error = 0.5 * (A_1 - y) ** 2
        dE_dZ = (A_1 - y) * activation_deriv(Z_1)
        dE_dW = tf.matmul(dE_dZ, tf.reshape(x, [1, -1]))

        print(f"Error: {Error.numpy()[0][0]:.4f}")
        print(f"dE_dW (Gradients): {np.round(dE_dW.numpy(), 4)}")
```

```python
        # Update weights
        W_0.assign_sub(lr * dE_dW)
        print(f"Updated W_0: {np.round(W_0.numpy(), 4)}")
```

```
Initial W_0 shape: (1, 3)
X shape: (2, 3)

EPOCH-1
================================================================================

ITER-1
--------------------------------------------------------------------------------
Target (y): 1.00
Z_1 (Weighted sum): 1.0000
A_1 (Activated output): 1.0000
Error: 0.0000
dE_dW (Gradients): [[0. 0. 0.]]
Updated W_0: [[1. 0. 1.]]

ITER-2
--------------------------------------------------------------------------------
Target (y): -1.00
Z_1 (Weighted sum): 2.0000
A_1 (Activated output): 2.0000
Error: 4.5000
dE_dW (Gradients): [[3. 3. 3.]]
Updated W_0: [[ 0.7 -0.3  0.7]]

ITER-3
--------------------------------------------------------------------------------
Target (y): -1.00
Z_1 (Weighted sum): 1.7000
A_1 (Activated output): 1.7000
Error: 3.6450
dE_dW (Gradients): [[ 2.7 -2.7  2.7]]
Updated W_0: [[ 0.43 -0.03  0.43]]
```

22CE1261 of cat vs dog

File Edit View Insert Runtime Tools Help

Files: dogs_vs_cats, sample_data, test, train, cat.jpg, dog.jpg, dogs-vs-cats.zip, kaggle.json

```
[1] !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/

[2] !kaggle datasets download -d salader/dogs-vs-cats

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Dataset URL: https://www.kaggle.com/datasets/salader/dogs-vs-cats
License(s): unknown

[3] import zipfile
    zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')
    zip_ref.extractall('/content')
    zip_ref.close()

[4] import tensorflow as tf
    from tensorflow import keras
    from keras import Sequential
    from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, BatchNormalization, Dropout, Input, SeparableConv2D, GlobalAveragePooling2D # Added Input, SeparableConv2D, and Gl
```

22CE1261 of cat vs dog

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

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

Found 20000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.

[6] 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)
```

22CE1261 of cat vs dog

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense, Dropout, BatchNormalization

model = Sequential()

# Conv Block 1
model.add(Conv2D(40, kernel_size=(3,3), activation='relu', padding='same', input_shape=(256, 256, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))

# Conv Block 2
model.add(Conv2D(72, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))

# Conv Block 3
model.add(Conv2D(144, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))

# Conv Block 4
model.add(Conv2D(220, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))

# Conv Block 5 (newly added for depth)
model.add(Conv2D(280, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))

# Global Pooling instead of Flatten
model.add(GlobalAveragePooling2D())

# Bigger Dense layer + more Dropout for regularization
model.add(Dense(96, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Connected to Python 3 Google Compute Engine backend (GPU)

CO 22CE1261 of cat vs dog

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_con
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 40)	1,120
batch_normalization (BatchNormalization)	(None, 256, 256, 40)	160
max_pooling2d (MaxPooling2D)	(None, 128, 128, 40)	0
conv2d_1 (Conv2D)	(None, 128, 128, 72)	25,992
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 72)	288
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 72)	0
conv2d_2 (Conv2D)	(None, 64, 64, 144)	93,456
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 144)	576
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 144)	0
conv2d_3 (Conv2D)	(None, 32, 32, 220)	285,340
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 220)	880
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 220)	0
conv2d_4 (Conv2D)	(None, 16, 16, 280)	554,680
batch_normalization_4 (BatchNormalization)	(None, 16, 16, 280)	1,120
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 280)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 280)	0
dense (Dense)	(None, 96)	26,976
dropout (Dropout)	(None, 96)	0
dense_1 (Dense)	(None, 1)	97

```
Total params: 990,685 (3.78 MB)
Trainable params: 989,173 (3.77 MB)
Non-trainable params: 1,512 (5.91 KB)
```

CO 22CE1261 of cat vs dog

File Edit View Insert Runtime Tools Help

Commands + Code + Text

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

```
history= model.fit(train_ds,epochs=15,validation_data=validation_ds)
```

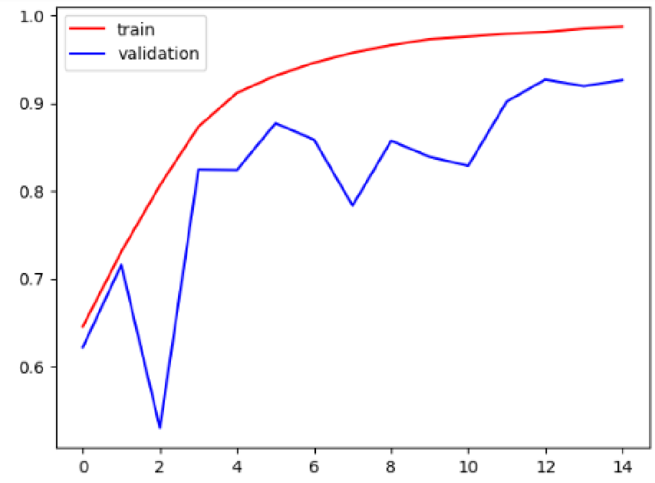
```
Epoch 1/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 98s 132ms/step - accuracy: 0.6086 - loss: 0.6955 - val_accuracy: 0.6216 - val_loss: 0.6908
Epoch 2/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 121s 118ms/step - accuracy: 0.7172 - loss: 0.5547 - val_accuracy: 0.7156 - val_loss: 0.6006
Epoch 3/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 80s 115ms/step - accuracy: 0.7874 - loss: 0.4601 - val_accuracy: 0.5298 - val_loss: 1.5325
Epoch 4/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 113ms/step - accuracy: 0.8648 - loss: 0.3143 - val_accuracy: 0.8244 - val_loss: 0.3770
Epoch 5/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 113ms/step - accuracy: 0.9073 - loss: 0.2286 - val_accuracy: 0.8238 - val_loss: 0.3751
Epoch 6/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 113ms/step - accuracy: 0.9282 - loss: 0.1798 - val_accuracy: 0.8772 - val_loss: 0.2834
Epoch 7/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - accuracy: 0.9445 - loss: 0.1435 - val_accuracy: 0.8584 - val_loss: 0.3372
Epoch 8/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - accuracy: 0.9565 - loss: 0.1116 - val_accuracy: 0.7832 - val_loss: 0.6871
Epoch 9/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 75s 120ms/step - accuracy: 0.9660 - loss: 0.0919 - val_accuracy: 0.8572 - val_loss: 0.4465
Epoch 10/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 78s 113ms/step - accuracy: 0.9743 - loss: 0.0685 - val_accuracy: 0.8388 - val_loss: 0.4892
Epoch 11/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 75s 119ms/step - accuracy: 0.9759 - loss: 0.0656 - val_accuracy: 0.8286 - val_loss: 0.5898
Epoch 12/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 79s 114ms/step - accuracy: 0.9796 - loss: 0.0528 - val_accuracy: 0.9020 - val_loss: 0.2678
Epoch 13/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 114ms/step - accuracy: 0.9805 - loss: 0.0514 - val_accuracy: 0.9270 - val_loss: 0.1962
Epoch 14/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 86s 120ms/step - accuracy: 0.9842 - loss: 0.0452 - val_accuracy: 0.9194 - val_loss: 0.2767
Epoch 15/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 77s 113ms/step - accuracy: 0.9879 - loss: 0.0343 - val_accuracy: 0.9264 - val_loss: 0.2602
```

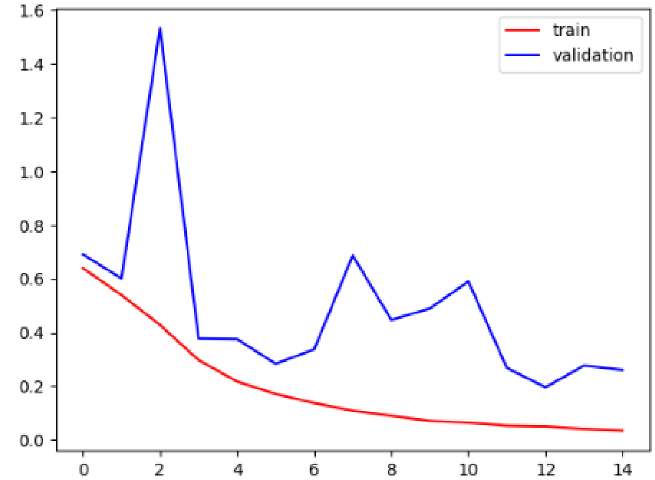
```
[10] import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()

plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

22CE1261 of cat vs dog

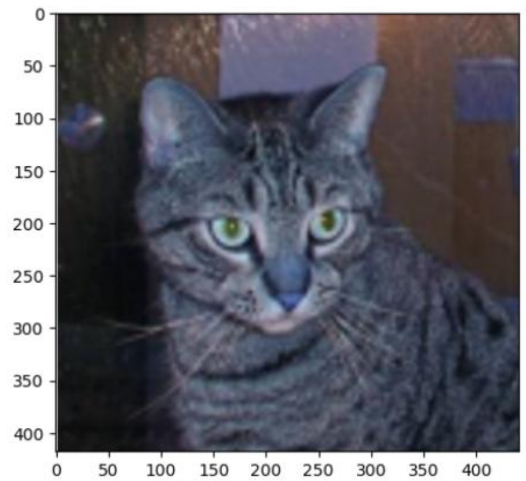
File Edit View Insert Runtime Tools Help

Commands + Code + Text



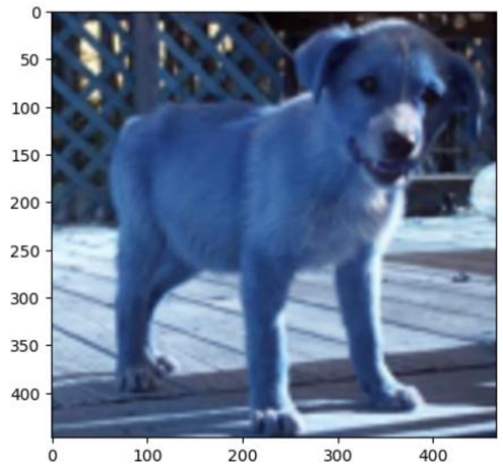



```
import cv2
test_img=cv2.imread('/content/cat.jpg')
plt.imshow(test_img)
plt.show()
test_img.shape
test_img=cv2.resize(test_img,(256,256))
test_input=test_img.reshape((1,256,256,3))
model.predict(test_input)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
array([[0.]], dtype=float32)
```

```
import cv2
test_img=cv2.imread('/content/dog.jpg')
plt.imshow(test_img)
plt.show()
test_img.shape
test_img=cv2.resize(test_img,(256,256))
test_input=test_img.reshape((1,256,256,3))
model.predict(test_input)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
array([[1.]], dtype=float32)
```