TensorFlow

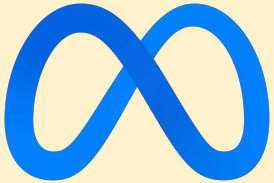
CS344
Deep Learning

WELLESLEY

Deep Learning Frameworks

TensorFlow

PyTorch



- Open source libraries
- Built on tensors
- Automatic differentiation
- High-level Keras interface

Create Model

Dense
(MLP, fully-connected)
layer

```
import tensorflow as tf
```

Two hidden layers

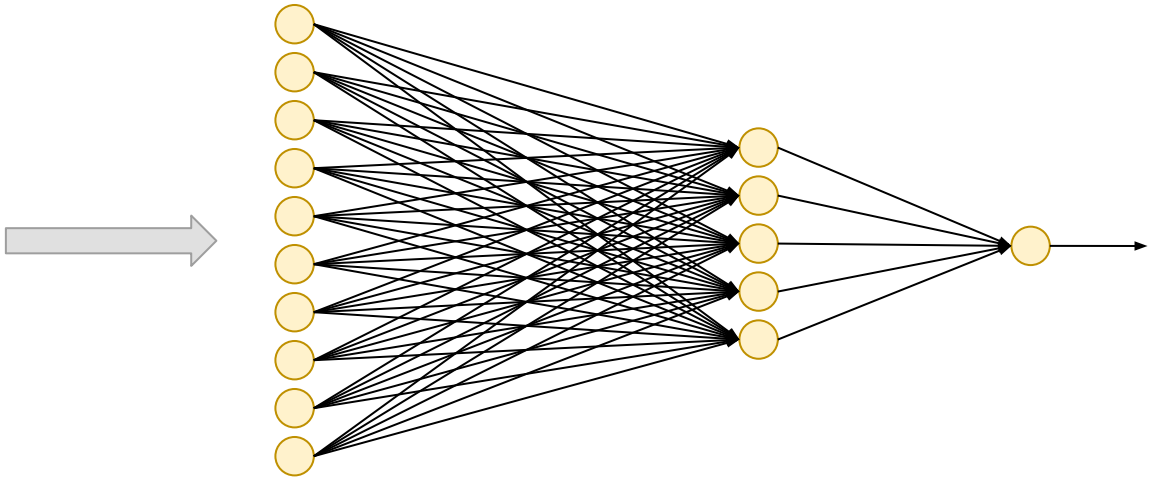
Activation function

```
# Create model
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(5, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Output layer

Units in layer

Architecture



Compile Model

Advanced gradient descent

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

Automatic Differentiation

Why?

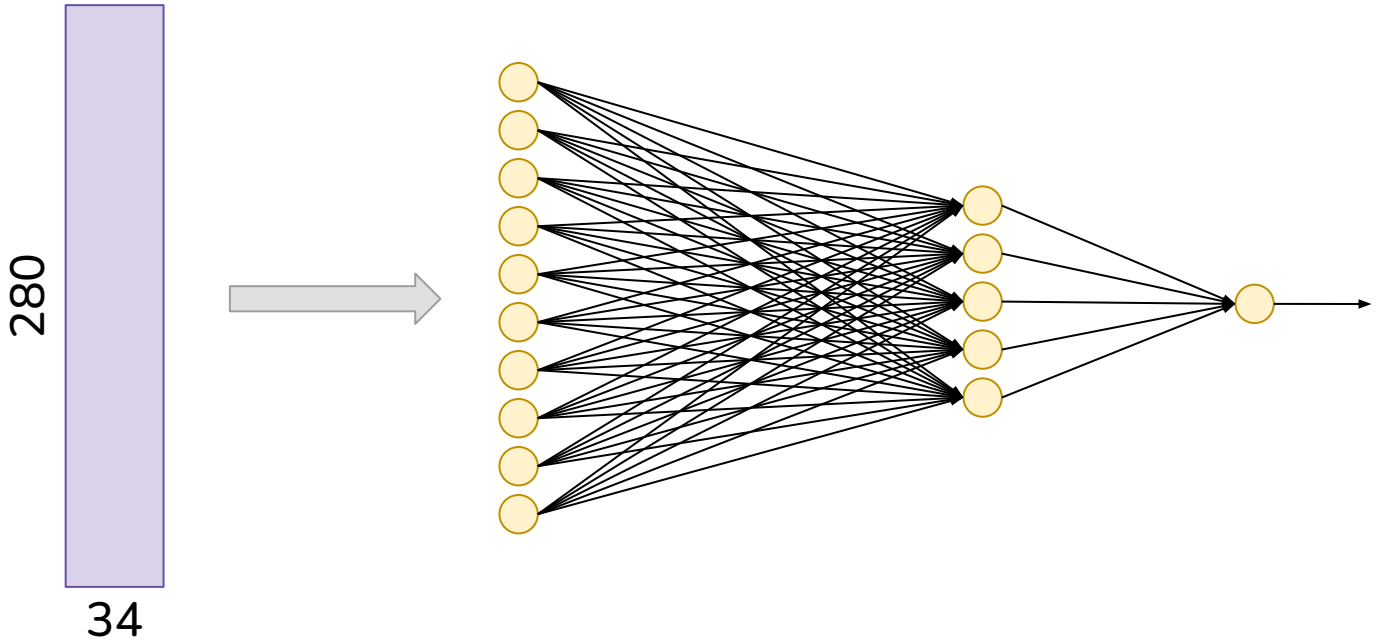
Train Model

Training data
X_train.shape is (280, 34)

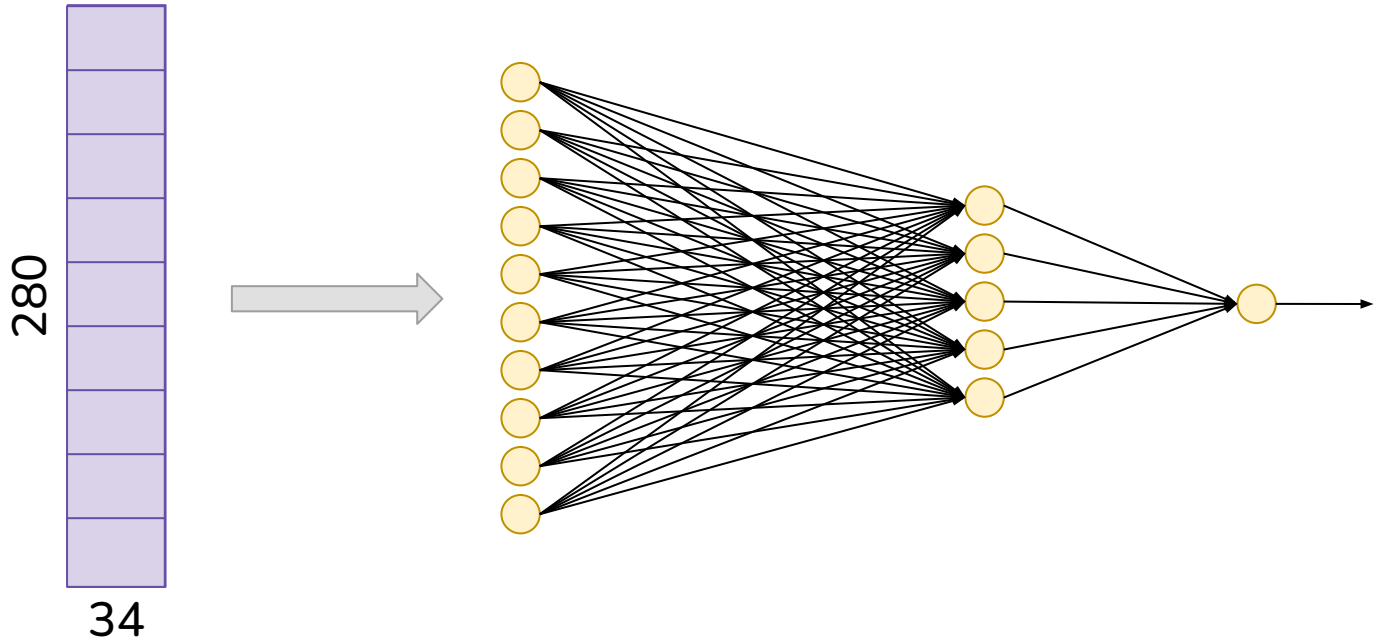
```
# Train model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=5, batch_size=32)
```

```
Epoch 1/5
9/9 [==============================] - 1s 13ms/step - loss: 0.7190 - accuracy: 0.4214 - val_loss: 0.7158 - val_accu
racy: 0.4366
Epoch 2/5
9/9 [==============================] - 0s 3ms/step - loss: 0.7039 - accuracy: 0.5679 - val_loss: 0.7035 - val_accur
acy: 0.5352
Epoch 3/5
9/9 [==============================] - 0s 3ms/step - loss: 0.6914 - accuracy: 0.6214 - val_loss: 0.6926 - val_accur
acy: 0.6197
Epoch 4/5
9/9 [==============================] - 0s 3ms/step - loss: 0.6812 - accuracy: 0.6857 - val_loss: 0.6831 - val_accur
acy: 0.6338
Epoch 5/5
9/9 [==============================] - 0s 3ms/step - loss: 0.6714 - accuracy: 0.7357 - val_loss: 0.6737 - val_accur
acy: 0.6901
```

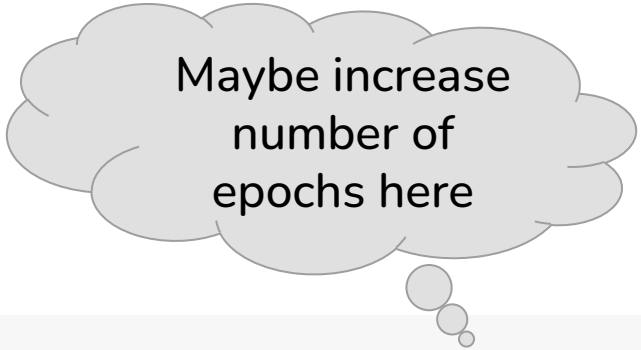
Batches per Epoch



Batches per Epoch



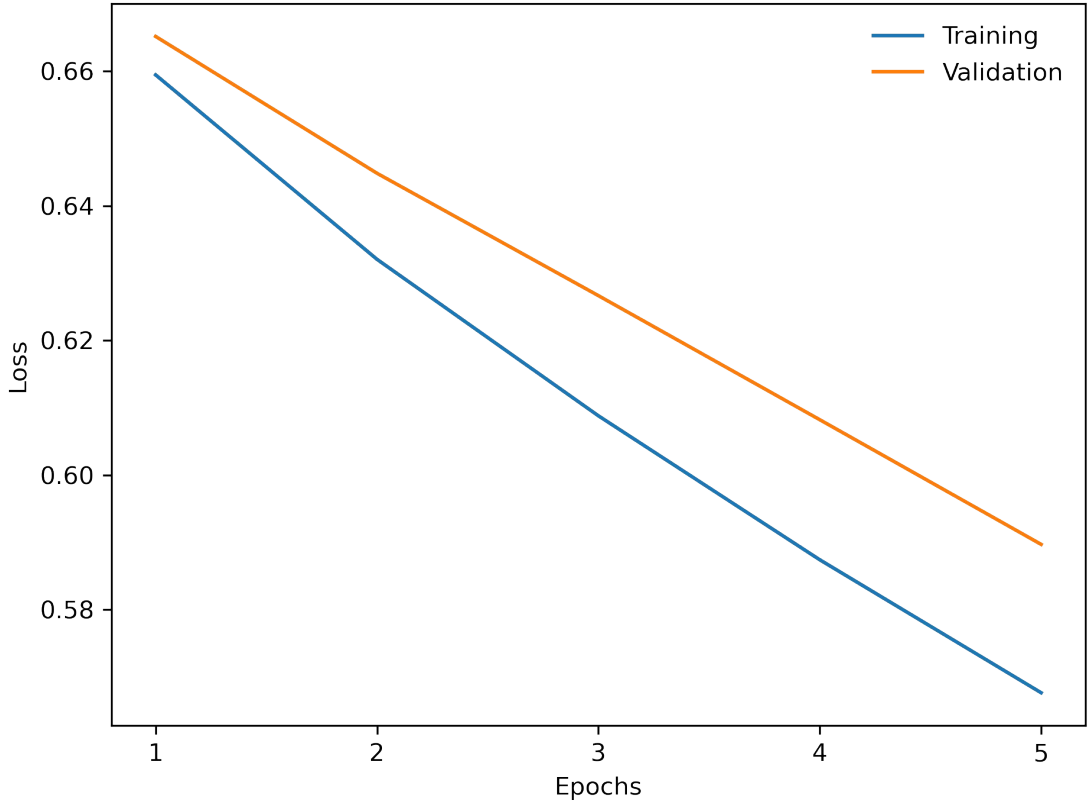
Train Model



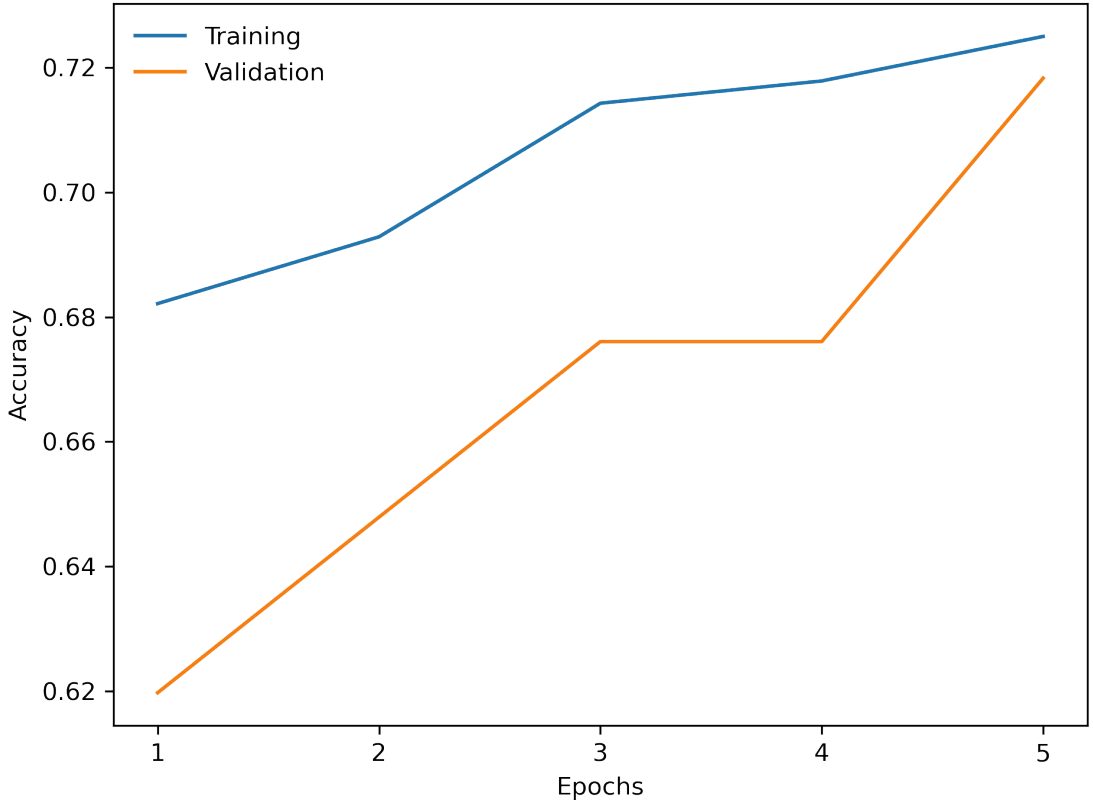

```
# Train model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=5, batch_size=32)
```

```
Epoch 1/5
9/9 [==============================] - 1s 13ms/step - loss: 0.7190 - accuracy: 0.4214 - val_loss: 0.7158 - val_accu
racy: 0.4366
Epoch 2/5
9/9 [==============================] - 0s 3ms/step - loss: 0.7039 - accuracy: 0.5679 - val_loss: 0.7035 - val_accur
acy: 0.5352
Epoch 3/5
9/9 [==============================] - 0s 3ms/step - loss: 0.6914 - accuracy: 0.6214 - val_loss: 0.6926 - val_accur
acy: 0.6197
Epoch 4/5
9/9 [==============================] - 0s 3ms/step - loss: 0.6812 - accuracy: 0.6857 - val_loss: 0.6831 - val_accur
acy: 0.6338
Epoch 5/5
9/9 [==============================] - 0s 3ms/step - loss: 0.6714 - accuracy: 0.7357 - val_loss: 0.6737 - val_accur
acy: 0.6901
```

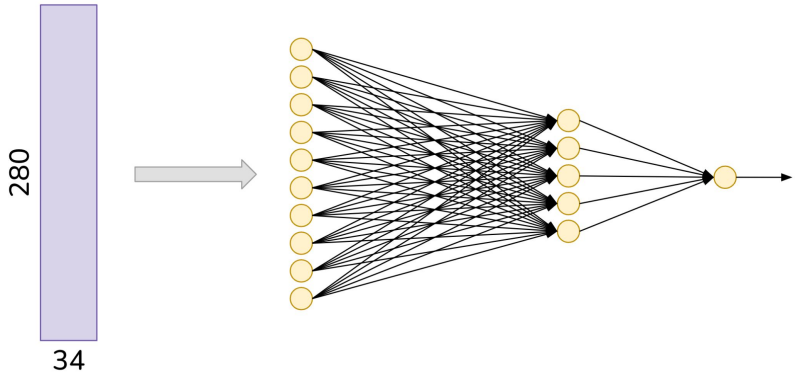
Performance During Training



Performance During Training



Model Info




```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense (Dense)                (None, 10)                350

 dense_1 (Dense)              (None, 5)                 55

 dense_2 (Dense)              (None, 1)                 6

=================================================================
Total params: 411
Trainable params: 411
Non-trainable params: 0
```

F1 Score

```
from sklearn.metrics import f1_score, classification_report
y_pred1 = model.predict(X_val)
if (len(class_names) <= 2): y_pred = np.round(y_pred1) # Binary
else: y_pred = np.argmax(y_pred1, axis=1)  # Multiclass
print('F1 score: ', f1_score(y_val, y_pred, average="weighted"))
print(classification_report(y_val, y_pred, target_names=class_names))
```

```
F1 score:  0.5188236336256533
              precision    recall  f1-score   support

     Class 0       0.67      0.13      0.22        30
     Class 1       0.60      0.95      0.74        41

    accuracy                           0.61        71
   macro avg       0.63      0.54      0.48        71
weighted avg       0.63      0.61      0.52        71
```

Save and Load model

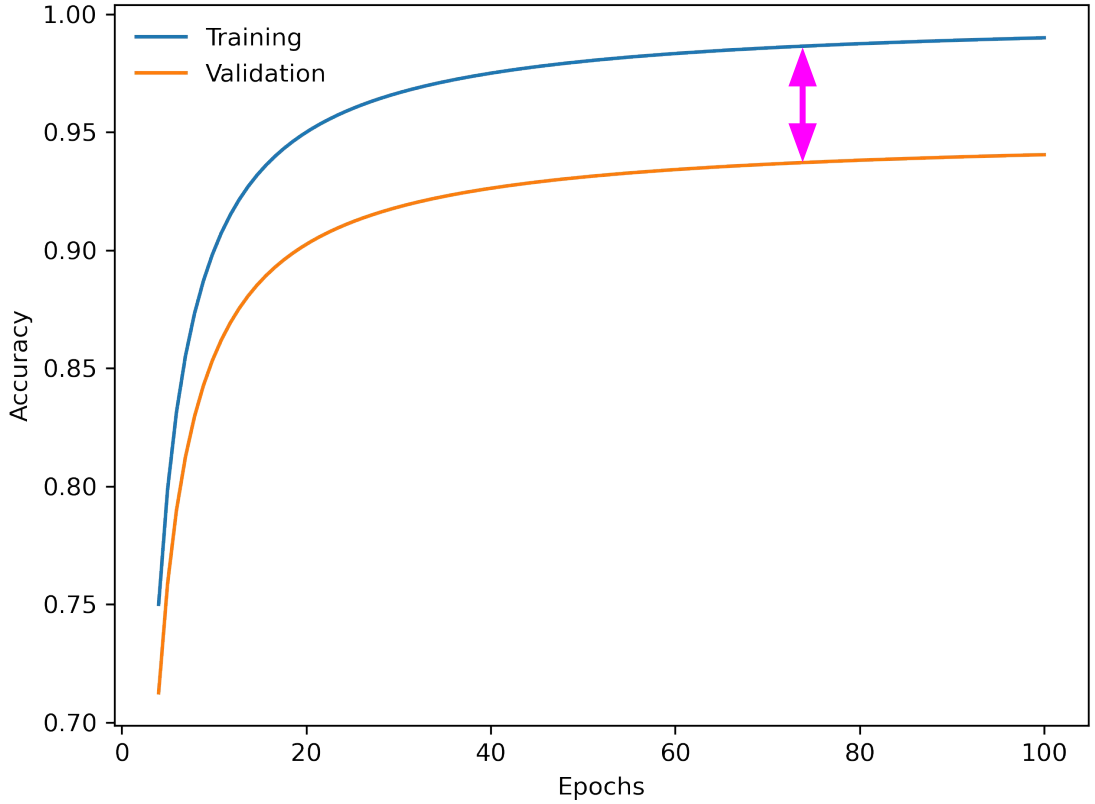
```
model.save('my_model.keras')
```

Save model to file

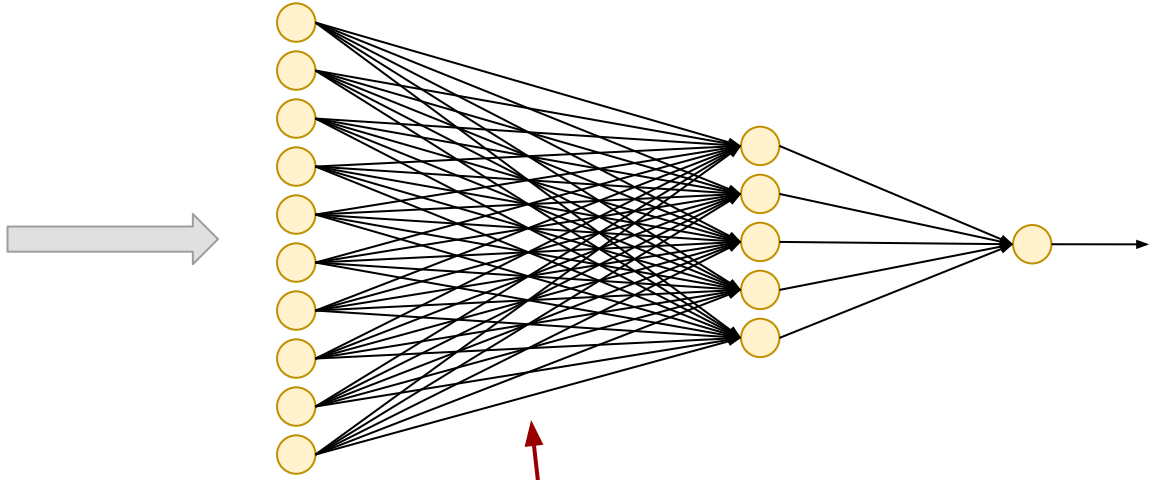
```
model = tf.keras.models.load_model('my_model.keras')
```

Load model from file

Overfitting

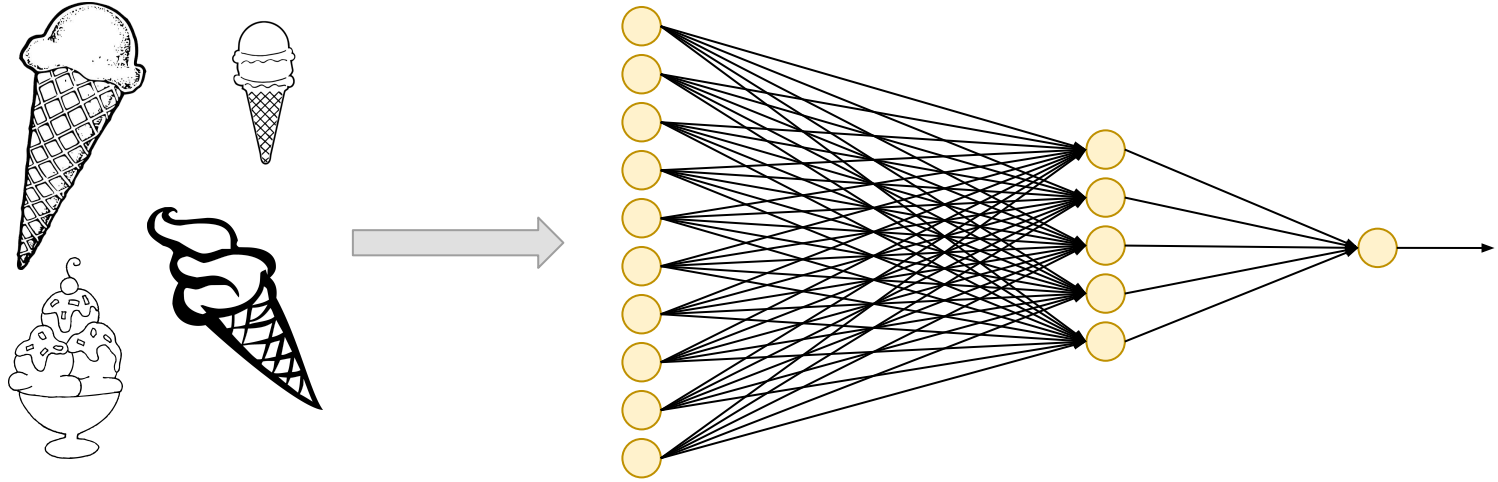


Dropout Regularization



```
# Create model
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(5, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

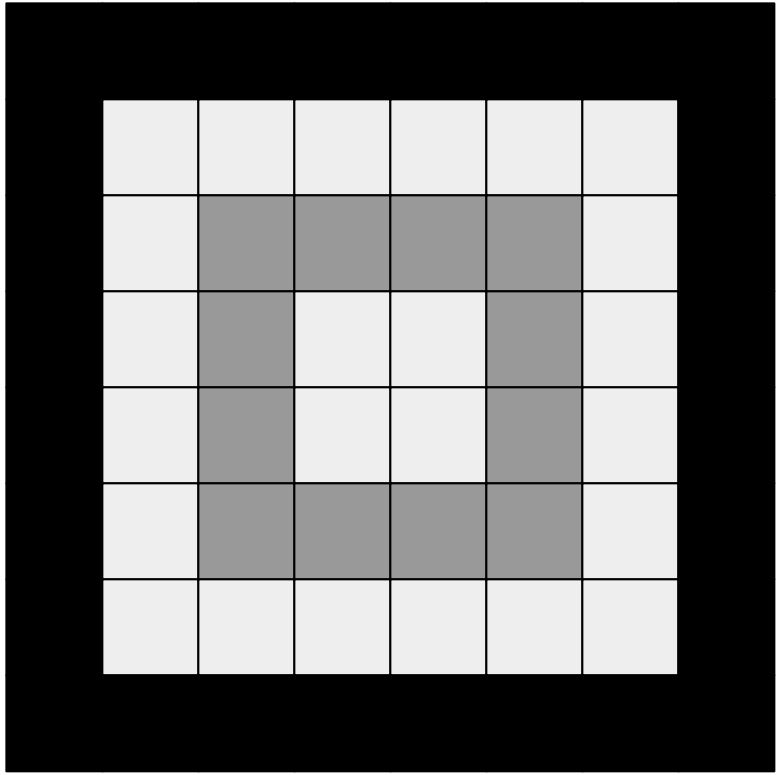
Preprocessing Layers

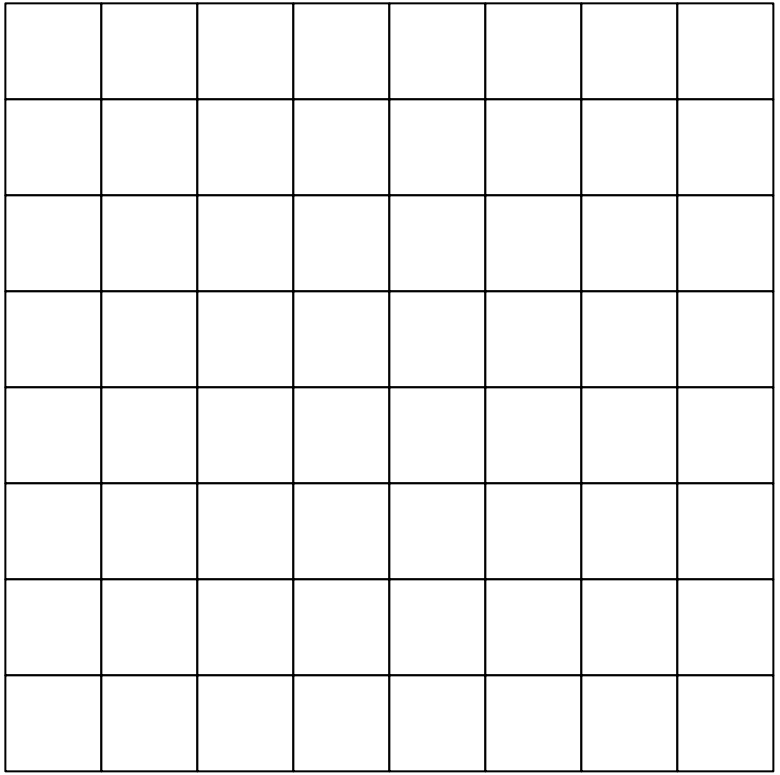


```
# Create model
model = tf.keras.Sequential([
  tf.keras.layers.Resizing(128, 128),
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(5, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

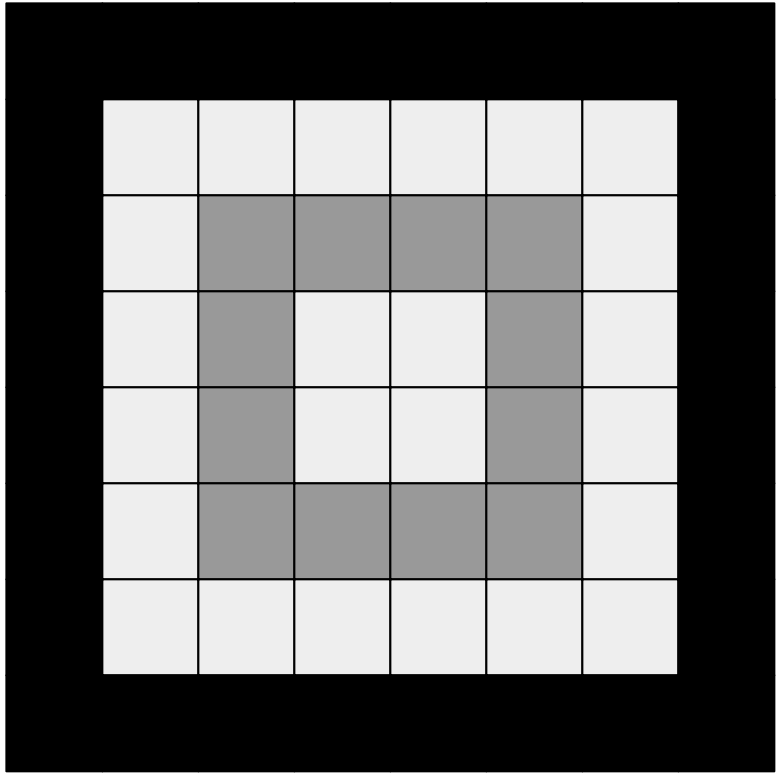
Images

Images as Pixels (Grayscale)



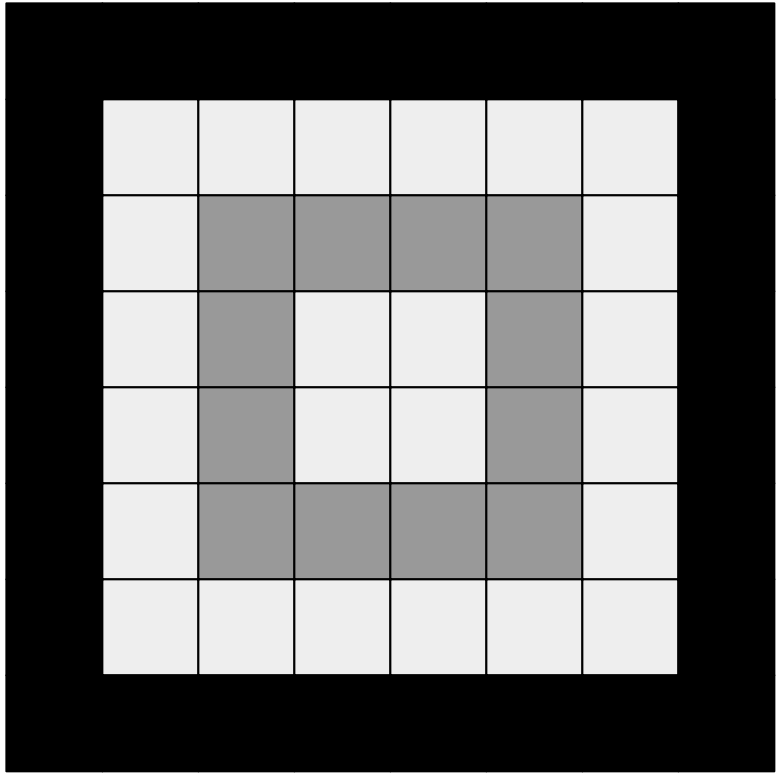


Images as Pixels (Grayscale)



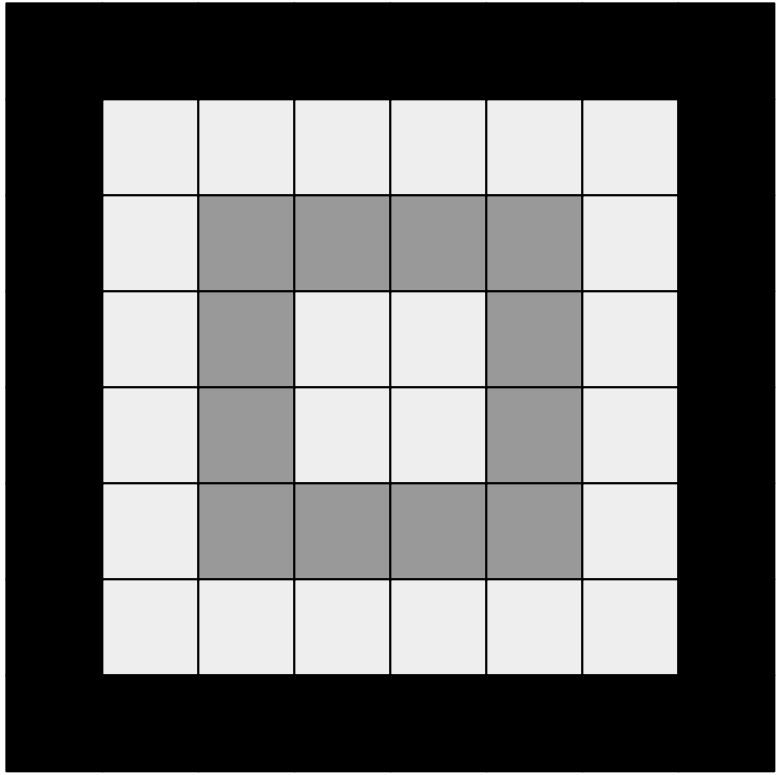
0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

Images as Pixels (Grayscale)



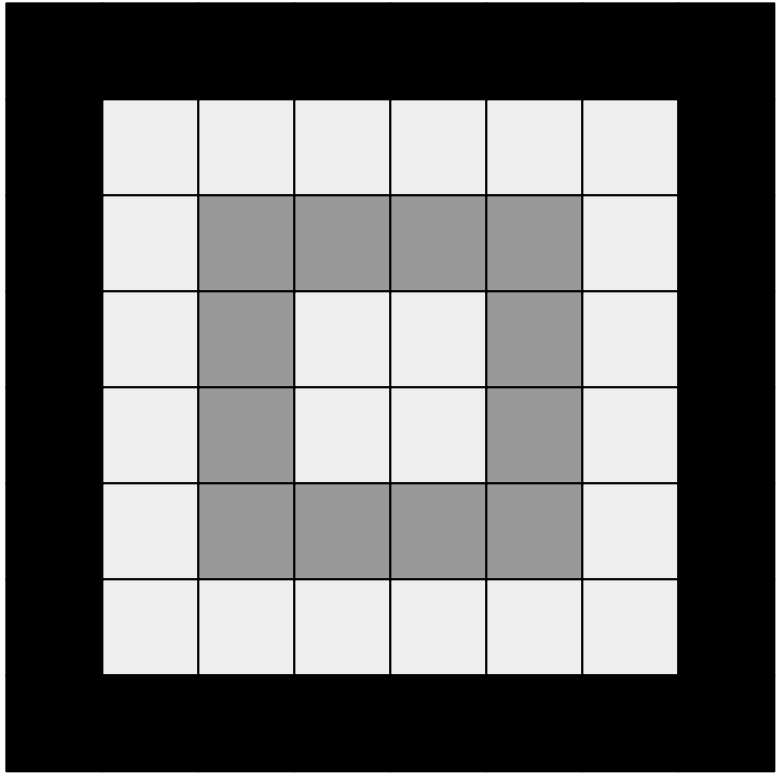
	255	255	255	255	255	255	
	255					255	
	255		255	255		255	
	255		255	255		255	
	255					255	
	255	255	255	255	255	255	

Images as Pixels (Grayscale)



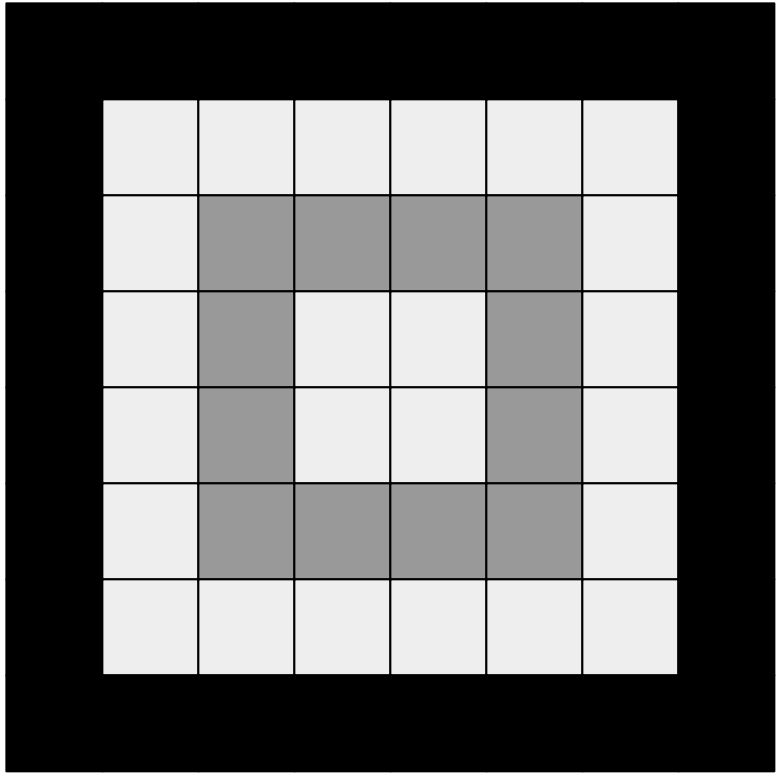
		128	128	128	128		
		128			128		
		128			128		
		128	128	128	128		

Images as Pixels (Grayscale)



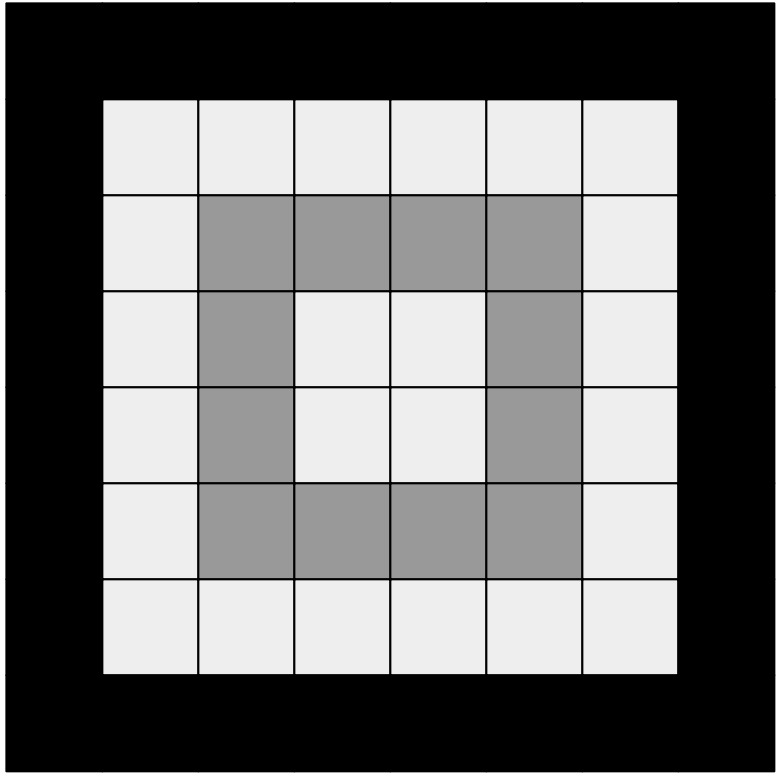
0	0	0	0	0	0	0	0
0	255	255	255	255	255	255	0
0	255	128	128	128	128	255	0
0	255	128	255	255	128	255	0
0	255	128	255	255	128	255	0
0	255	128	128	128	128	255	0
0	255	255	255	255	255	255	0
0	0	0	0	0	0	0	0

Images as Pixels (Grayscale)



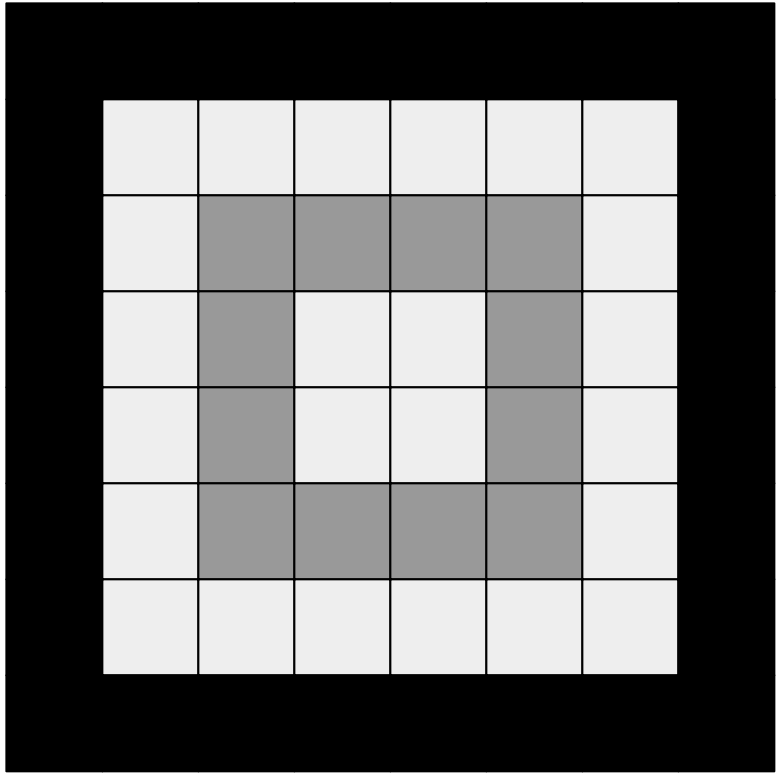
	1	1	1	1	1	1	
	1					1	
	1		1	1		1	
	1		1	1		1	
	1					1	
	1	1	1	1	1	1	

Images as Pixels (Grayscale)



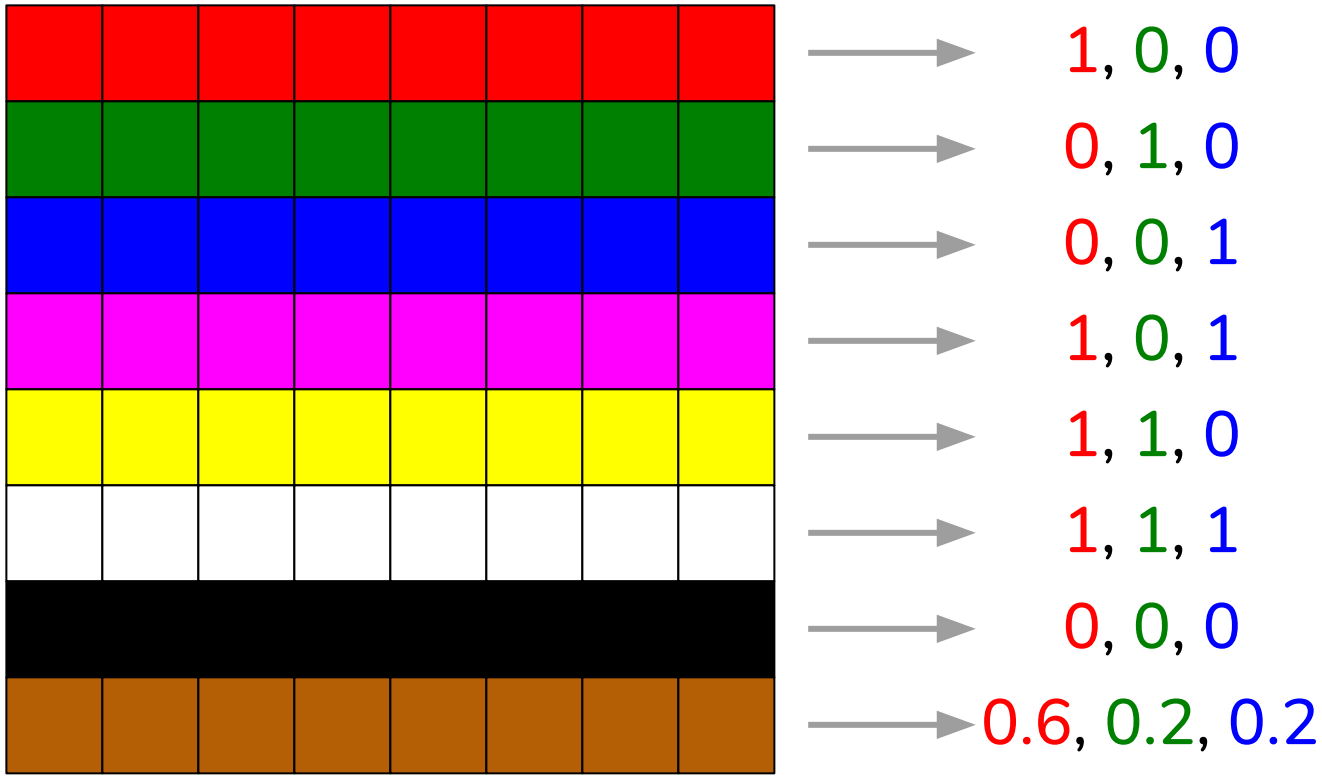
0	0	0	0	0	0	0	0
0							0
0		0.5	0.5	0.5	0.5		0
0		0.5			0.5		0
0		0.5			0.5		0
0		0.5	0.5	0.5	0.5		0
0							0
0	0	0	0	0	0	0	0

Images as Pixels (Grayscale)

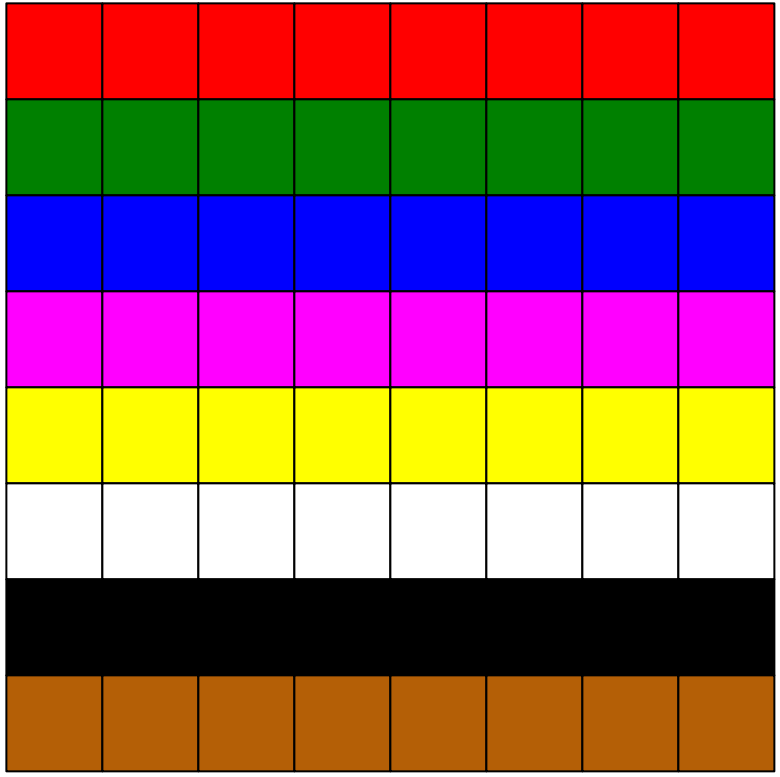


0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	0.5	0.5	0.5	0.5	1	0
0	1	0.5	1	1	0.5	1	0
0	1	0.5	1	1	0.5	1	0
0	1	0.5	0.5	0.5	0.5	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

Images as Pixels (RGB)



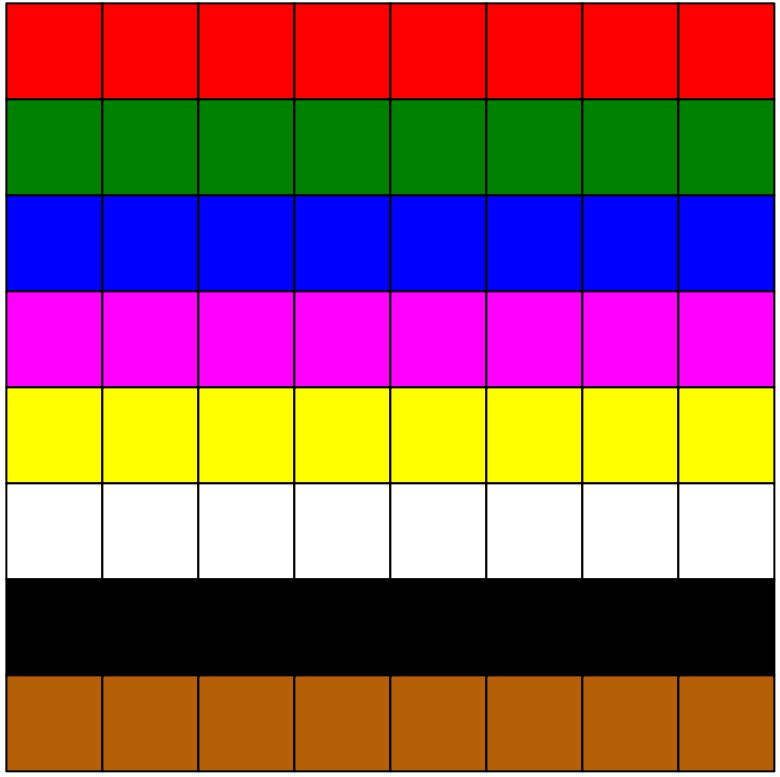
Images as Pixels (RGB)



Red Channel

1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6

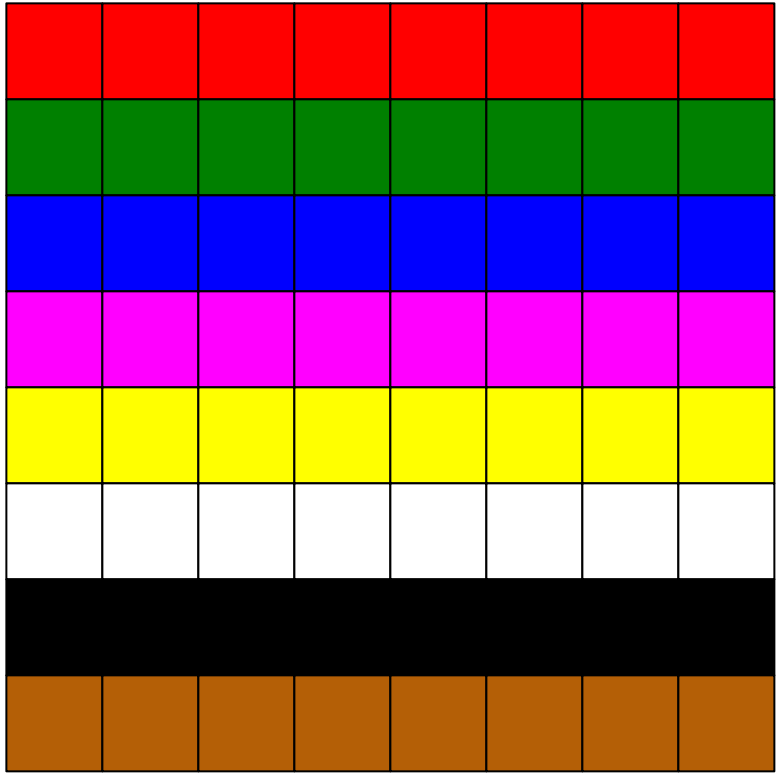
Images as Pixels (RGB)



Green Channel

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2

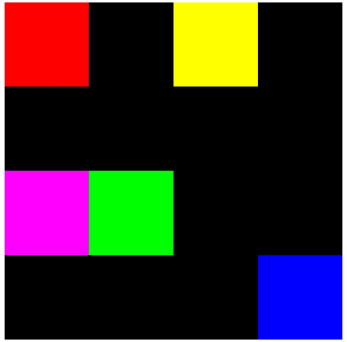
Images as Pixels (RGB)



Blue Channel

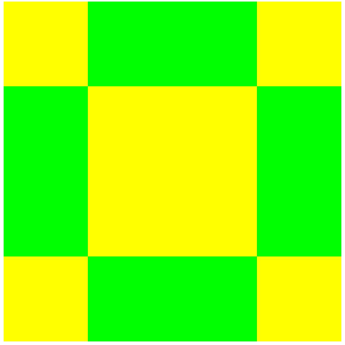
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2

Images as Pixels (RGB)



```
np.array([[[255, 0, 0],   [0, 0, 0],   [255, 255, 0], [0, 0, 0]],
          [[0, 0, 0],     [0, 0, 0],   [0, 0, 0],     [0, 0, 0]],
          [[255, 0, 255], [0, 255, 0], [0, 0, 0],     [0, 0, 0]],
          [[0, 0, 0],     [0, 0, 0],   [0, 0, 0],     [0, 0, 255]]])
```

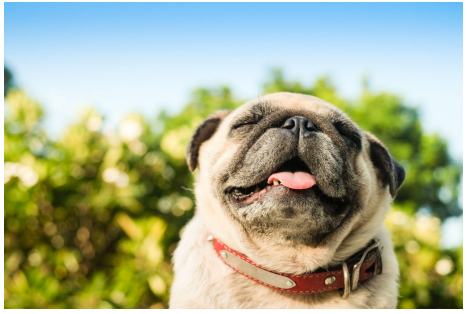
(4, 4, 3)



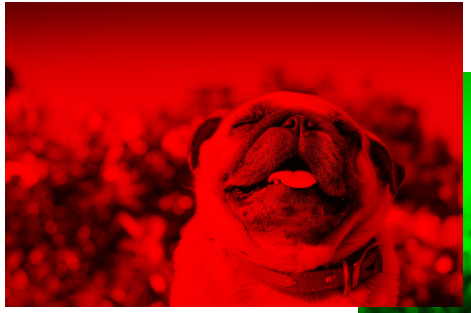
```
np.array([[[255, 255, 0], [0, 255, 0],   [0, 255, 0],   [255, 255, 0]],
          [[0, 255, 0],   [255, 255, 0], [255, 255, 0], [0, 255, 0]],
          [[0, 255, 0],   [255, 255, 0], [255, 255, 0], [0, 255, 0]],
          [[255, 255, 0], [0, 255, 0],   [0, 255, 0],   [255, 255, 0]]])
```

Images as Pixels (RGB)

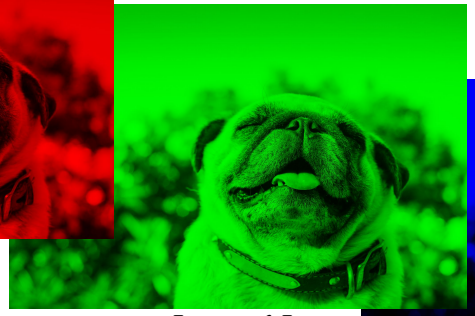
p



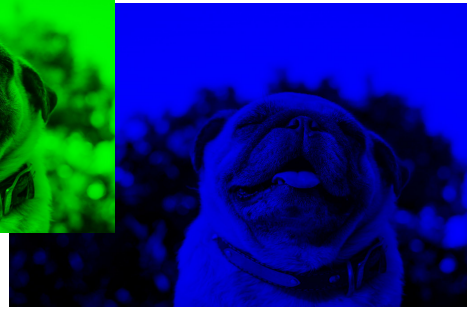
(3264, 4896, 3)



p[:, :, 0]



p[:, :, 1]

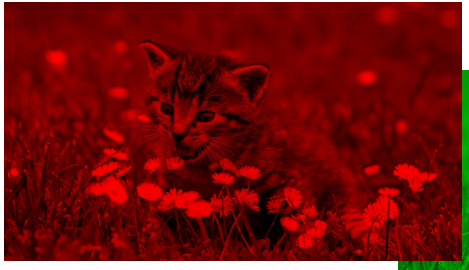


p[:, :, 2]

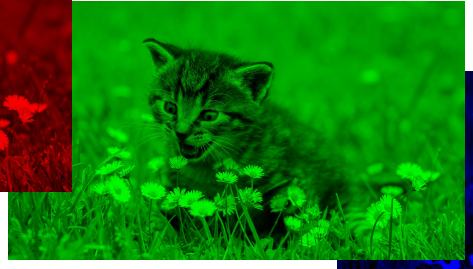
k

(1440, 2560, 3)

k[:, :, 0]



k[:, :, 1]



k[:, :, 2]

Images with MLPs

k

(1440, 2560, 3)

k.flatten()

(11059200,)

How many parameters if we didn't resize?
How many parameters if we did resize?