```
from google.colab import drive
from os import listdir
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
#from keras.layers.normalization import BatchNormalization
from keras.layers import Activation, Flatten, Dense, Dropout
#from tensorflow.keras import layers
from keras import layers
import tensorflow as tf
from tensorflow import keras
# plot dog photos from the dogs vs cats dataset
from matplotlib import pyplot as plt
from matplotlib.image import imread
```

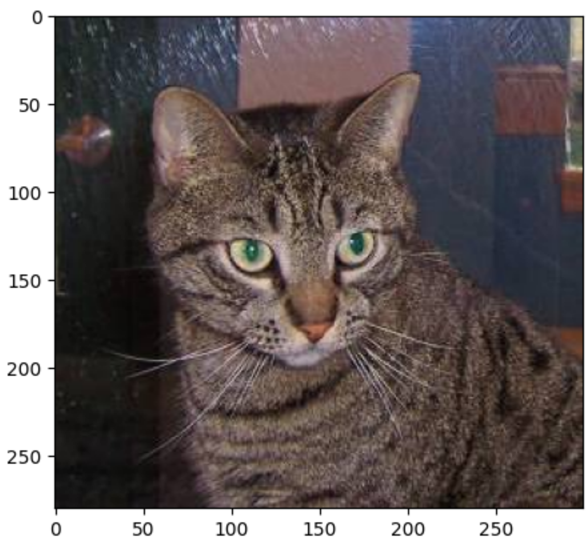
```
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Professor's notebook

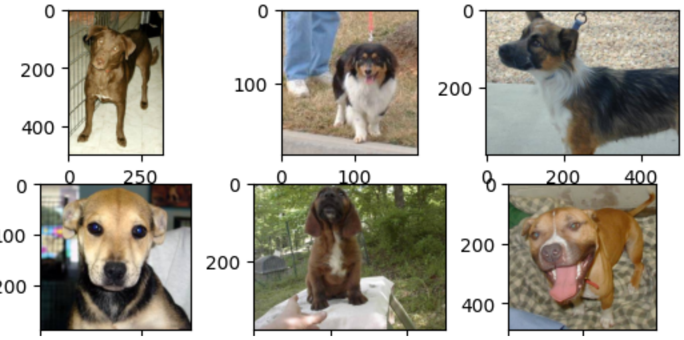
```
#import cv2 as cv
folder = 'gdrive/MyDrive/dogcat/dogvscat1000'
img = imread(folder + '/cat.1.jpg')
plt.imshow(img)
#link = "https://drive.google.com/drive/folders/1h3YimZqnkkoz1fADS5WjuR86aG0jP2Ql?usp=sharing"
```

```
<matplotlib.image.AxesImage at 0x7bec07502c50>
```

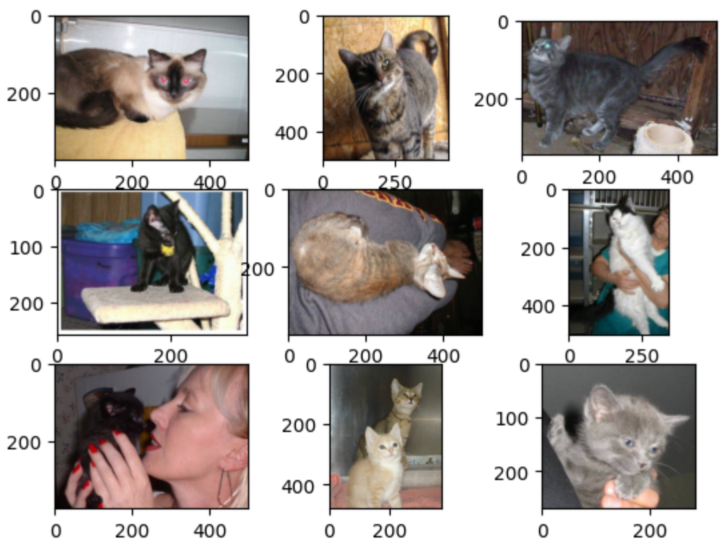



```
def displayImages(foldername,dogorcat,startID):
    # plot first few images
    for i in range(9):
        #define subplot 3x3
        plt.subplot(330 + 1 + i)
        # define filename
        filename = foldername + '/' + dogorcat +'.' + str(i+startID) + '.jpg'
        # load image pixels
        image = imread(filename)
        # plot raw pixel data
        plt.imshow(image)
        # show the figure
plt.show()
```

```
displayImages(folder,"dog",1)
```



```
displayImages(folder,"cat",20)
```



```
# define location of dataset
#folder = 'dogvscat1000/'
photos, labels = list(), list()
# enumerate files in the directory
for file in listdir(folder):
    # determine class
    output = 0.0
    if file.startswith('cat'):
        output = 1.0
        # load image
    photo = load_img(folder + '/' + file, target_size=(32, 32),color_mode="rgb")  #"rgb" for color mode; "grayscale"
    # convert to numpy array
    photo = img_to_array(photo)
    # store
    photos.append(photo)
    labels.append(output)
# convert to a numpy arrays
photos = asarray(photos)
labels = asarray(labels)
print(photos.shape, labels.shape)
# save the reshaped photos
save('dogs_vs_cats_photos.npy', photos)
save('dogs_vs_cats_labels.npy', labels)
```

```
(1000, 32, 32, 3) (1000,)
```

```
print(photos[1].shape)
```

```
(32, 32, 3)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(photos, labels, test_size=.4, random_state=42)
```

```
# declare a model
model = keras.Sequential([
    keras.layers.LayerNormalization(axis=3 , center=True , scale=True),
    keras.layers.Flatten(input_shape=(32, 32, 3),name="Input"),
    keras.layers.Dense(256, activation='relu',name="Hidden"),
    keras.layers.Dropout(rate=0.5),
    keras.layers.BatchNormalization(),
```

```
    keras.layers.Dense(2, name="Output"),
])
```

```
# another way to declair a model
model_alt = keras.Sequential()
model_alt.add(layers.LayerNormalization(axis=3 , center=True , scale=True))
model_alt.add(layers.Flatten(input_shape=(32, 32, 3),name="Input"))
model_alt.add(layers.Dense(256, activation='relu',name="Hidden"))
model_alt.add(layers.Dropout(rate=0.5))
model_alt.add(layers.BatchNormalization())
model_alt.add(layers.Dense(2, name="Output"))
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model_alt.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
history=model.fit(X_train, y_train, epochs=100,verbose=1,batch_size=100)
history=model_alt.fit(X_train, y_train, epochs=100,verbose=1,batch_size=100)
```

```
Epoch 72/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0269 - accuracy: 0.9967
Epoch 73/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0206 - accuracy: 0.9950
Epoch 74/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0258 - accuracy: 0.9917
Epoch 75/100
6/6 [==============================] - 0s 34ms/step - loss: 0.0374 - accuracy: 0.9917
Epoch 76/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0269 - accuracy: 0.9917
Epoch 77/100
6/6 [==============================] - 0s 31ms/step - loss: 0.0214 - accuracy: 0.9967
Epoch 78/100
6/6 [==============================] - 0s 33ms/step - loss: 0.0205 - accuracy: 0.9933
Epoch 79/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0149 - accuracy: 0.9983
Epoch 80/100
6/6 [==============================] - 0s 31ms/step - loss: 0.0167 - accuracy: 0.9967
Epoch 81/100
6/6 [==============================] - 0s 33ms/step - loss: 0.0159 - accuracy: 0.9983
Epoch 82/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0134 - accuracy: 1.0000
Epoch 83/100
6/6 [==============================] - 0s 33ms/step - loss: 0.0115 - accuracy: 0.9983
Epoch 84/100
6/6 [==============================] - 0s 31ms/step - loss: 0.0104 - accuracy: 0.9983
Epoch 85/100
6/6 [==============================] - 0s 31ms/step - loss: 0.0056 - accuracy: 1.0000
Epoch 86/100
6/6 [==============================] - 0s 33ms/step - loss: 0.0060 - accuracy: 1.0000
Epoch 87/100
6/6 [==============================] - 0s 31ms/step - loss: 0.0058 - accuracy: 0.9983
Epoch 88/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0069 - accuracy: 0.9983
Epoch 89/100
6/6 [==============================] - 0s 34ms/step - loss: 0.0115 - accuracy: 0.9983
Epoch 90/100
6/6 [==============================] - 0s 44ms/step - loss: 0.0063 - accuracy: 1.0000
Epoch 91/100
6/6 [==============================] - 0s 51ms/step - loss: 0.0058 - accuracy: 0.9983
Epoch 92/100
6/6 [==============================] - 0s 51ms/step - loss: 0.0071 - accuracy: 0.9967
Epoch 93/100
6/6 [==============================] - 0s 47ms/step - loss: 0.0064 - accuracy: 0.9983
Epoch 94/100
6/6 [==============================] - 0s 46ms/step - loss: 0.0045 - accuracy: 1.0000
Epoch 95/100
6/6 [==============================] - 0s 51ms/step - loss: 0.0041 - accuracy: 1.0000
Epoch 96/100
6/6 [==============================] - 0s 45ms/step - loss: 0.0034 - accuracy: 1.0000
Epoch 97/100
6/6 [==============================] - 0s 44ms/step - loss: 0.0072 - accuracy: 0.9983
Epoch 98/100
6/6 [==============================] - 0s 45ms/step - loss: 0.0057 - accuracy: 0.9983
Epoch 99/100
6/6 [==============================] - 0s 46ms/step - loss: 0.0039 - accuracy: 1.0000
Epoch 100/100
6/6 [==============================] - 0s 45ms/step - loss: 0.0043 - accuracy: 1.0000
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 layer_normalization (Layer  (100, 32, 32, 3)          6
 Normalization)

 Input (Flatten)             (100, 3072)               0

 Hidden (Dense)              (100, 256)                786688

 dropout (Dropout)           (100, 256)                0

 batch_normalization (Batch  (100, 256)                1024
 Normalization)

 Output (Dense)              (100, 2)                  514

=================================================================
Total params: 788232 (3.01 MB)
Trainable params: 787720 (3.00 MB)
Non-trainable params: 512 (2.00 KB)
_________________________________________________________________
```

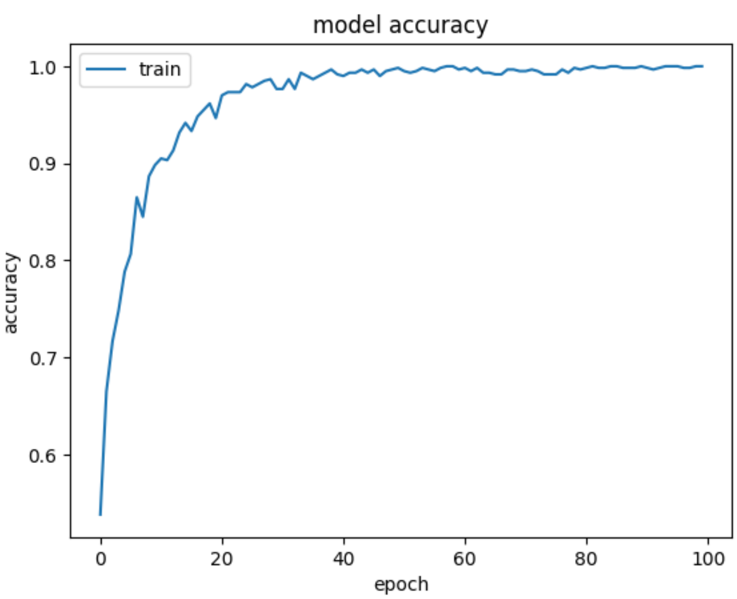
```
test_loss, test_acc = model.evaluate(X_test,  y_test, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
13/13 - 0s - loss: 1.5515 - accuracy: 0.5950 - 372ms/epoch - 29ms/step

Test accuracy: 0.5950000286102295
```

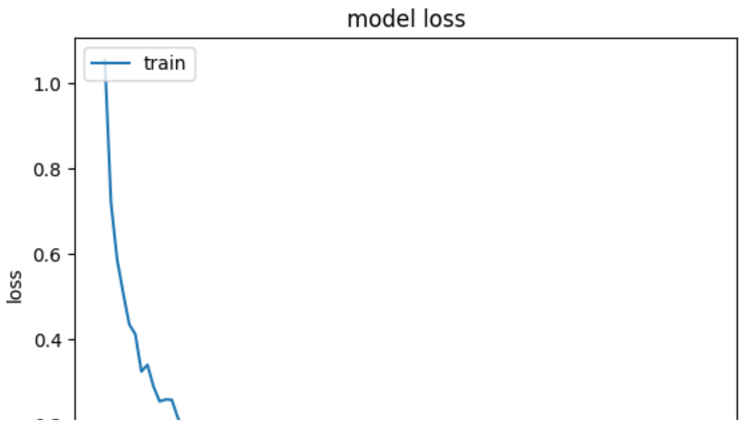
```
# list all data in history
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy'])
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```




```
# summarize history for loss
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



K-FOLD 1

```python
from sklearn.model_selection import KFold
import numpy as np
```

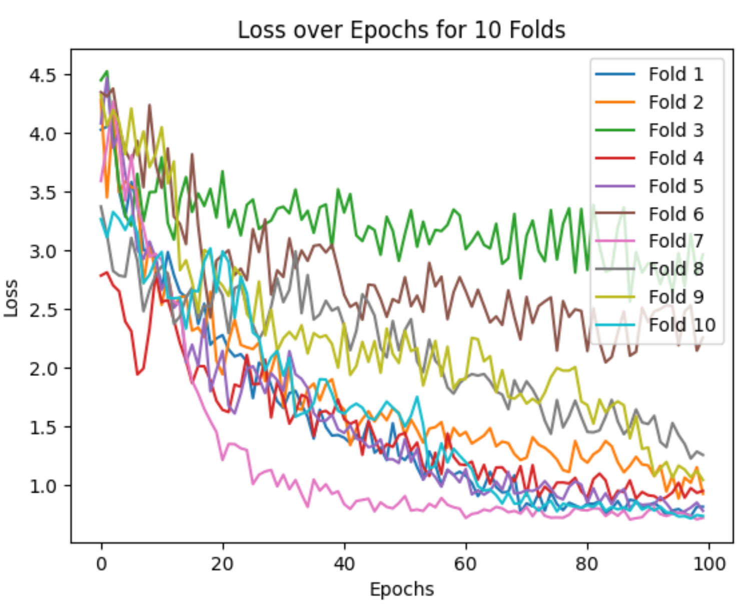
```python
model = tf.keras.Sequential([
    keras.layers.Flatten(input_shape=(32, 32, 3),name="Input"),
    tf.keras.layers.Dense(64, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax'),
    keras.layers.Dropout(rate=0.5),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(2, name="Output"),
])
```

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

```python
n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

loss_history = []
accuracy_scores = []

epochs = 100

for train_index, test_index in kf.split(photos):
    x_train, x_test = photos[train_index], photos[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    # Reset the model for each fold
    model = tf.keras.Sequential([
        keras.layers.Flatten(input_shape=(32, 32, 3),name="Input"),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dropout(rate=0.5),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(2, name="Output"),
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Training loop
    history = model.fit(x_train, y_train, epochs=epochs, validation_data=(x_test, y_test), verbose=1)

    # Append the test accuracy to the list
    accuracy_scores.append(history.history['val_accuracy'][-1])

    # Store loss history for later plotting
    loss_history.append(history.history['loss'])


```

```
29/29 [==============================] - 0s 11ms/step - loss: 3.3252 - accuracy: 0.4767 - val_loss: 1.9723 - val_accuracy: 0.4700
Epoch 4/100
29/29 [==============================] - 0s 11ms/step - loss: 3.2652 - accuracy: 0.4978 - val_loss: 1.8935 - val_accuracy: 0.5400
Epoch 5/100
29/29 [==============================] - 0s 12ms/step - loss: 3.1722 - accuracy: 0.5000 - val_loss: 4.7654 - val_accuracy: 0.4800
Epoch 6/100
29/29 [==============================] - 0s 10ms/step - loss: 3.2882 - accuracy: 0.4667 - val_loss: 0.9986 - val_accuracy: 0.4800
Epoch 7/100
29/29 [==============================] - 0s 8ms/step - loss: 3.1458 - accuracy: 0.4956 - val_loss: 1.3469 - val_accuracy: 0.5100
Epoch 8/100
29/29 [==============================] - 0s 9ms/step - loss: 2.7187 - accuracy: 0.4889 - val_loss: 2.4882 - val_accuracy: 0.5200
Epoch 9/100
29/29 [==============================] - 0s 9ms/step - loss: 2.7912 - accuracy: 0.5222 - val_loss: 4.0072 - val_accuracy: 0.5500
Epoch 10/100
29/29 [==============================] - 0s 8ms/step - loss: 2.9208 - accuracy: 0.4878 - val_loss: 2.2113 - val_accuracy: 0.4000
Epoch 11/100
29/29 [==============================] - 0s 9ms/step - loss: 2.9850 - accuracy: 0.5000 - val_loss: 2.1092 - val_accuracy: 0.5500
Epoch 12/100
29/29 [==============================] - 0s 9ms/step - loss: 2.5958 - accuracy: 0.5100 - val_loss: 1.1467 - val_accuracy: 0.5500
Epoch 13/100
29/29 [==============================] - 0s 9ms/step - loss: 2.5901 - accuracy: 0.5011 - val_loss: 0.6807 - val_accuracy: 0.5500
Epoch 14/100
29/29 [==============================] - 0s 9ms/step - loss: 2.5961 - accuracy: 0.4956 - val_loss: 0.6931 - val_accuracy: 0.5300
Epoch 15/100
29/29 [==============================] - 0s 9ms/step - loss: 2.3319 - accuracy: 0.5078 - val_loss: 0.6931 - val_accuracy: 0.5100
Epoch 16/100
29/29 [==============================] - 0s 9ms/step - loss: 2.6562 - accuracy: 0.5233 - val_loss: 5.3226 - val_accuracy: 0.4500
Epoch 17/100
29/29 [==============================] - 0s 9ms/step - loss: 2.6493 - accuracy: 0.5178 - val_loss: 3.7799 - val_accuracy: 0.4500
Epoch 18/100
29/29 [==============================] - 0s 9ms/step - loss: 2.9053 - accuracy: 0.5278 - val_loss: 1.6497 - val_accuracy: 0.4400
Epoch 19/100
29/29 [==============================] - 0s 8ms/step - loss: 3.0159 - accuracy: 0.4878 - val_loss: 1.0518 - val_accuracy: 0.4700
Epoch 20/100
29/29 [==============================] - 0s 8ms/step - loss: 2.6428 - accuracy: 0.4933 - val_loss: 1.0765 - val_accuracy: 0.4700
Epoch 21/100
29/29 [==============================] - 0s 9ms/step - loss: 2.9901 - accuracy: 0.5211 - val_loss: 1.0709 - val_accuracy: 0.4700
Epoch 22/100
29/29 [==============================] - 0s 10ms/step - loss: 2.8987 - accuracy: 0.5133 - val_loss: 0.9405 - val_accuracy: 0.4500
Epoch 23/100
29/29 [==============================] - 0s 10ms/step - loss: 2.4235 - accuracy: 0.5389 - val_loss: 1.0457 - val_accuracy: 0.4400
Epoch 24/100
29/29 [==============================] - 0s 8ms/step - loss: 2.7738 - accuracy: 0.4767 - val_loss: 0.7952 - val_accuracy: 0.4500
Epoch 25/100
29/29 [==============================] - 0s 8ms/step - loss: 2.6320 - accuracy: 0.5200 - val_loss: 0.6862 - val_accuracy: 0.4300
Epoch 26/100
29/29 [==============================] - 0s 9ms/step - loss: 2.2984 - accuracy: 0.5222 - val_loss: 0.9247 - val_accuracy: 0.4500
Epoch 27/100
29/29 [==============================] - 0s 8ms/step - loss: 2.2547 - accuracy: 0.5222 - val_loss: 0.9138 - val_accuracy: 0.4400
Epoch 28/100
29/29 [==============================] - 0s 10ms/step - loss: 1.9768 - accuracy: 0.5244 - val_loss: 0.6862 - val_accuracy: 0.4600
```

```python

# Plot the loss history
for fold, loss in enumerate(loss_history, start=1):
    plt.plot(range(epochs), loss, label=f"Fold {fold}")
plt.title("Loss over Epochs for 10 Folds")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```python
# Calculate and report mean and standard deviation of accuracy
mean_accuracy = np.mean(accuracy_scores)
std_accuracy = np.std(accuracy_scores)
print(f"Mean accuracy: {mean_accuracy}")
print(f"Standard deviation of accuracy: {std_accuracy}")
```

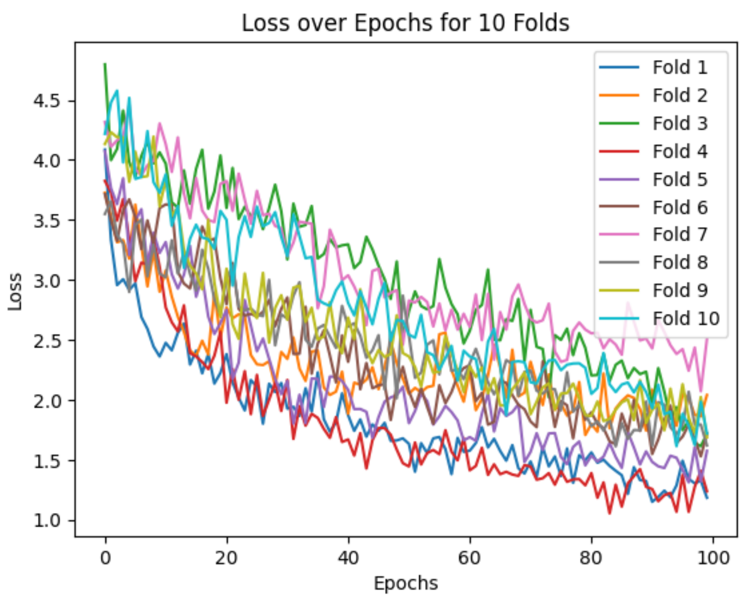
```
Mean accuracy: 0.47799999415874483
Standard deviation of accuracy: 0.06029924722647785
```

K-FOLD 2

```python
n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

loss_history_dn = []
accuracy_scores_dn = []

epochs = 100

for train_index, test_index in kf.split(photos):
    x_train, x_test = photos[train_index], photos[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    # Reset the model for each fold
    model = tf.keras.Sequential([
        keras.layers.Flatten(input_shape=(32, 32, 3),name="Input"),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dropout(rate=0.5),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dropout(rate=0.5),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(2, name="Output"),
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Training loop
    history = model.fit(x_train, y_train, epochs=epochs, validation_data=(x_test, y_test), verbose=1)

    # Append the test accuracy to the list
    accuracy_scores_dn.append(history.history['val_accuracy'][-1])

    # Store loss history for later plotting
    loss_history_dn.append(history.history['loss'])


```

```
29/29 [==============================] - 0s 11ms/step - loss: 2.0525 - accuracy: 0.5278 - val_loss: 2.2305 - val_accuracy: 0.5300
Epoch 65/100
29/29 [==============================] - 0s 12ms/step - loss: 2.1801 - accuracy: 0.4900 - val_loss: 1.7437 - val_accuracy: 0.5000
Epoch 66/100
29/29 [==============================] - 0s 11ms/step - loss: 1.9550 - accuracy: 0.5200 - val_loss: 1.6454 - val_accuracy: 0.5100
Epoch 67/100
29/29 [==============================] - 0s 9ms/step - loss: 1.9507 - accuracy: 0.5389 - val_loss: 1.0927 - val_accuracy: 0.5200
Epoch 68/100
29/29 [==============================] - 0s 9ms/step - loss: 2.0885 - accuracy: 0.4878 - val_loss: 1.0396 - val_accuracy: 0.5400
Epoch 69/100
29/29 [==============================] - 0s 9ms/step - loss: 2.1527 - accuracy: 0.4944 - val_loss: 0.9618 - val_accuracy: 0.5300
Epoch 70/100
29/29 [==============================] - 0s 9ms/step - loss: 2.2283 - accuracy: 0.4811 - val_loss: 1.1014 - val_accuracy: 0.5200
Epoch 71/100
29/29 [==============================] - 0s 8ms/step - loss: 1.8799 - accuracy: 0.4844 - val_loss: 1.8787 - val_accuracy: 0.5200
Epoch 72/100
29/29 [==============================] - 0s 9ms/step - loss: 1.9839 - accuracy: 0.4833 - val_loss: 1.6302 - val_accuracy: 0.5400
Epoch 73/100
29/29 [==============================] - 0s 9ms/step - loss: 2.1035 - accuracy: 0.4889 - val_loss: 1.4903 - val_accuracy: 0.5400
Epoch 74/100
29/29 [==============================] - 0s 9ms/step - loss: 2.1376 - accuracy: 0.5011 - val_loss: 1.4742 - val_accuracy: 0.5500
Epoch 75/100
29/29 [==============================] - 0s 8ms/step - loss: 1.9187 - accuracy: 0.5300 - val_loss: 2.6950 - val_accuracy: 0.5500
Epoch 76/100
```

```
# Plot the loss history
for fold, loss in enumerate(loss_history_dn, start=1):
    plt.plot(range(epochs), loss, label=f"Fold {fold}")
plt.title("Loss over Epochs for 10 Folds")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
# Calculate and report mean and standard deviation of accuracy
mean_accuracy_dn = np.mean(accuracy_scores_dn)
std_accuracy_dn = np.std(accuracy_scores_dn)
print(f"Mean accuracy: {mean_accuracy_dn}")
print(f"Standard deviation of accuracy: {std_accuracy_dn}")
```

```
Mean accuracy: 0.5189999997615814
Standard deviation of accuracy: 0.05185556724294938
```

```
# Compare the means and standard deviations
mean_difference = mean_accuracy_dn - mean_accuracy
std_accuracy_difference = std_accuracy_dn - std_accuracy

print(f"Mean accuracy difference: {mean_difference}")
print(f"Standard deviation of accuracy difference: {std_accuracy_difference}")


```

```
Mean accuracy difference: 0.041000005602836564
Standard deviation of accuracy difference: -0.00844367998352847
```

In the first k-fold network I added a drop out layer and batch normalization after the last dense layer.

In the second k-fold network I added a drop out layer and batch normalization after each dense layer.

The second version had a higher mean accuracy and a lower standard deviation.