

MUHAMMAD HUZAIFA OWAIS

Understanding AI (771763_C23_T3A)

Summative Assignment:

Exercise 2: Image Recognition to Identify Species of Flowers

In [1]:

```
import tensorflow as tf
import os
import numpy as np
from PIL import Image
from tensorflow.keras.preprocessing.image import img_to_array
```

Loading the Data

In [2]:

```
def load_flower_dataset(dataset_path):
    data = []
    labels = []
    class_names = os.listdir(dataset_path)
    class_indices = {class_name: idx for idx, class_name in enumerate(class_names)}

    for class_name in class_names:
        class_path = os.path.join(dataset_path, class_name)
        if not os.path.isdir(class_path):
            continue
        for file in os.listdir(class_path):
            if file.endswith(('jpg', 'jpeg', 'png')):
                file_path = os.path.join(class_path, file)
                try:
                    img = Image.open(file_path).convert('RGB')
                    img = img.resize((128, 128)) # Resize to a fixed size
                    img_array = img_to_array(img)
                    data.append(img_array)
                    labels.append(class_indices[class_name])
                except Exception as e:
                    print(f"Error loading image {file_path}: {e}")

    data = np.array(data, dtype='float32') / 255.0 # Normalize images
    labels = np.array(labels)
    return data, labels, class_names

# Load the dataset
dataset_path = 'D:/UNIVERSITY OF HULL/UOH/Understanding AI Assignment/flower_photos'
data, labels, class_names = load_flower_dataset(dataset_path)
```

Splitting into Test, Train

In [3]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    data, labels, test_size=0.2, random_state=42 # 20% for test set
)
```

```
# Print shapes of the datasets
print(f"Training data shape: {x_train.shape}")
print(f"Testing data shape: {x_test.shape}")
print(f"Training labels shape: {y_train.shape}")
print(f"Testing labels shape: {y_test.shape}")
print(f"Class names: {class_names}")
```

```
Training data shape: (2936, 128, 128, 3)
Testing data shape: (734, 128, 128, 3)
Training labels shape: (2936,)
Testing labels shape: (734,)
Class names: ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

Label encoding to One hot Encoding

In [4]:

```
import keras
# Converting label encoding to one hot encoding
y_train_cat = keras.utils.to_categorical(y_train)
y_test_cat = keras.utils.to_categorical(y_test)
```

Data Augmentation

In [5]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rotation_range=20,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   horizontal_flip=True,
                                   vertical_flip=False,
                                   shear_range=0.10,
                                   zoom_range=0.10,
                                   validation_split=0.2)
```

CONVOLUTIONAL NEURAL NETWORK

Initial Model

Three convolutional layer with 32,64 & 128 filters. Kernel Size 3 x 3 and ReLU activation function

Max pooling of 2 x 2 after each conv layer

FC layer with 64 neurons

20 % dropout after FC layer

Output layer with softmax activation function

Batch Size 32

Adam Optimizer

20 Epochs

In [72]:

```
from tensorflow.keras.models import Sequential
```

```

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
model0 = Sequential()
# First Convolutional and pooling layer
model0.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model0.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model0.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model0.add(Flatten())
# Dense layer with 64 neurons
model0.add(Dense(64, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model0.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model0.add(Dense(5, activation = 'softmax'))

```

In [73]:

```

from keras.optimizers import Adam

model0.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history0 = model0.fit(train_datagen.flow(x_train, y_train_cat,
                                         batch_size = batch_size,
                                         subset = "training"),
                    epochs = 20, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation"))

```

Epoch 1/20

74/74 ————— 37s 424ms/step - accuracy: 0.2945 - loss: 1.5659 - val_accuracy: 0.4514 - val_loss: 1.2528

Epoch 2/20

74/74 ————— 29s 383ms/step - accuracy: 0.4799 - loss: 1.1868 - val_accuracy: 0.5434 - val_loss: 1.1040

Epoch 3/20

74/74 ————— 29s 382ms/step - accuracy: 0.5617 - loss: 1.0928 - val_accuracy: 0.6065 - val_loss: 1.0265

Epoch 4/20

74/74 ————— 29s 386ms/step - accuracy: 0.6072 - loss: 1.0295 - val_accuracy: 0.6491 - val_loss: 0.9905

Epoch 5/20

74/74 ————— 29s 382ms/step - accuracy: 0.6133 - loss: 0.9815 - val_accuracy: 0.6746 - val_loss: 0.8948

Epoch 6/20

74/74 ————— 29s 383ms/step - accuracy: 0.6413 - loss: 0.8969 - val_accuracy: 0.6576 - val_loss: 0.9322

Epoch 7/20

74/74 ————— 29s 383ms/step - accuracy: 0.6851 - loss: 0.8363 - val_accuracy: 0.6627 - val_loss: 0.8966

Epoch 8/20

74/74 ————— 30s 388ms/step - accuracy: 0.6559 - loss: 0.8272 - val_accuracy: 0.7104 - val_loss: 0.8195

Epoch 9/20

74/74 ————— 30s 386ms/step - accuracy: 0.6834 - loss: 0.7998 - val_accuracy: 0.6899 - val_loss: 0.8049

Epoch 10/20

74/74 ————— 30s 394ms/step - accuracy: 0.7189 - loss: 0.7543 - val_accuracy: 0.7070 - val_loss: 0.8488

Epoch 11/20

74/74 ————— 30s 388ms/step - accuracy: 0.7108 - loss: 0.7714 - val_accuracy: 0.7155 - val_loss: 0.7982

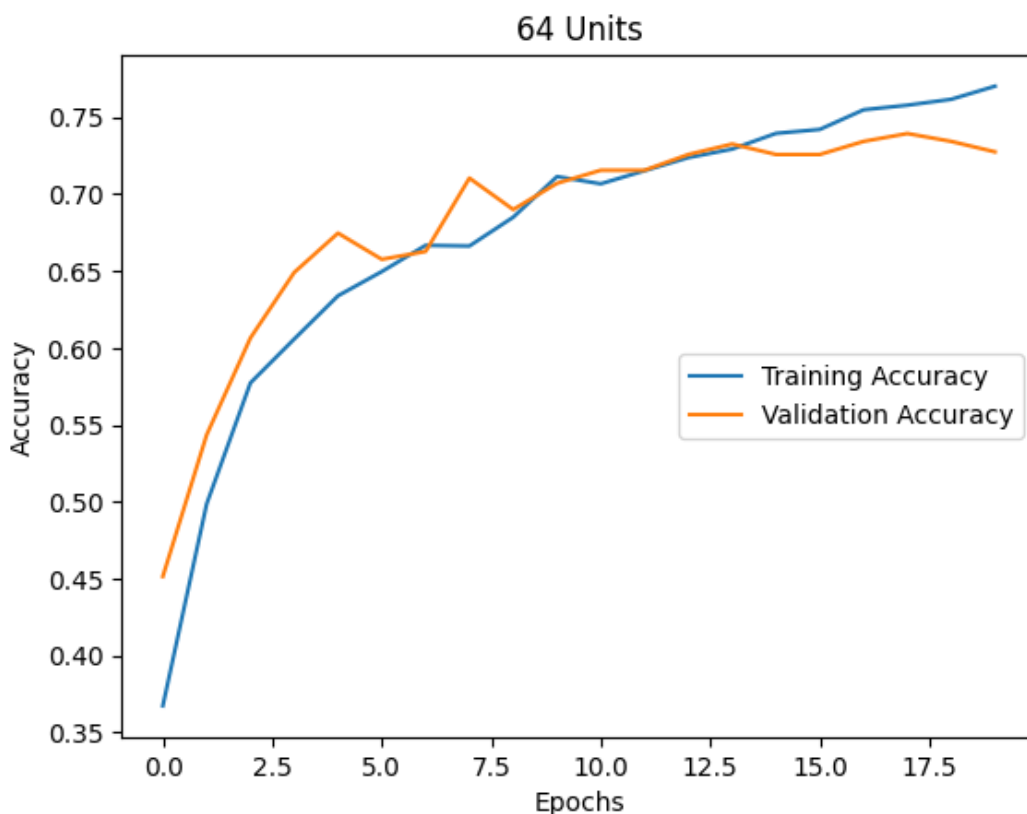
Epoch 12/20

74/74 ————— 30s 387ms/step - accuracy: 0.7090 - loss: 0.7434 - val_accuracy: 0.7155 - val_loss: 0.7771

Epoch 13/20
74/74 ————— 30s 389ms/step - accuracy: 0.7297 - loss: 0.6991 - val_accuracy: 0.7257 - val_loss: 0.7757
Epoch 14/20
74/74 ————— 29s 383ms/step - accuracy: 0.7476 - loss: 0.6642 - val_accuracy: 0.7325 - val_loss: 0.7434
Epoch 15/20
74/74 ————— 30s 387ms/step - accuracy: 0.7560 - loss: 0.6421 - val_accuracy: 0.7257 - val_loss: 0.7281
Epoch 16/20
74/74 ————— 31s 401ms/step - accuracy: 0.7582 - loss: 0.6370 - val_accuracy: 0.7257 - val_loss: 0.7181
Epoch 17/20
74/74 ————— 30s 401ms/step - accuracy: 0.7554 - loss: 0.6290 - val_accuracy: 0.7342 - val_loss: 0.7891
Epoch 18/20
74/74 ————— 30s 394ms/step - accuracy: 0.7583 - loss: 0.6254 - val_accuracy: 0.7394 - val_loss: 0.7086
Epoch 19/20
74/74 ————— 30s 396ms/step - accuracy: 0.7567 - loss: 0.6127 - val_accuracy: 0.7342 - val_loss: 0.7614
Epoch 20/20
74/74 ————— 30s 394ms/step - accuracy: 0.7768 - loss: 0.6068 - val_accuracy: 0.7274 - val_loss: 0.7350

In [138]:

```
import pandas as pd
import matplotlib.pyplot as plt
# Creating Dataframe
history_df = pd.DataFrame(history0.history)
#Plotting
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("64 Units")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("64 Units")
plt.show()
```



Hyperparameter Tunning

FC layer 128 units

In [83]:


```
model01 = Sequential()
# First Convolutional and pooling layer
model01.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model01.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model01.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model01.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model01.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model01.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model01.add(Flatten())
# Dense layer with 128 neurons
model01.add(Dense(128, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model01.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model01.add(Dense(5, activation = 'softmax'))
```

In [84]:


```
model01.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history01 = model01.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))
```


Epoch 1/20

74/74  37s 457ms/step - accuracy: 0.3364 - loss: 1.4450 - val_accuracy: 0.4600 - val_loss: 1.2583


Epoch 2/20

74/74  34s 446ms/step - accuracy: 0.5321 - loss: 1.1437 - val_accuracy: 0.6235 - val_loss: 1.0903


Epoch 3/20

74/74  34s 444ms/step - accuracy: 0.6339 - loss: 0.9686 - val_accuracy: 0.6644 - val_loss: 0.9042


Epoch 4/20

74/74  34s 447ms/step - accuracy: 0.6467 - loss: 0.9311 - val_accuracy: 0.6593 - val_loss: 0.9136


Epoch 5/20

74/74  35s 455ms/step - accuracy: 0.6556 - loss: 0.8756 - val_accuracy: 0.7002 - val_loss: 0.8408


Epoch 6/20

74/74  34s 445ms/step - accuracy: 0.7065 - loss: 0.7914 - val_accuracy: 0.6882 - val_loss: 0.8044


Epoch 7/20

74/74  34s 448ms/step - accuracy: 0.7083 - loss: 0.7283 - val_accuracy: 0.7087 - val_loss: 0.7771


Epoch 8/20

74/74  34s 449ms/step - accuracy: 0.7248 - loss: 0.7285 - val_accuracy: 0.7019 - val_loss: 0.7928


Epoch 9/20

74/74  34s 451ms/step - accuracy: 0.7471 - loss: 0.6693 - val_accuracy: 0.7019 - val_loss: 0.8002

Epoch 10/20

74/74  34s 443ms/step - accuracy: 0.7066 - loss: 0.7506 - val_accuracy: 0.7291 - val_loss: 0.7332

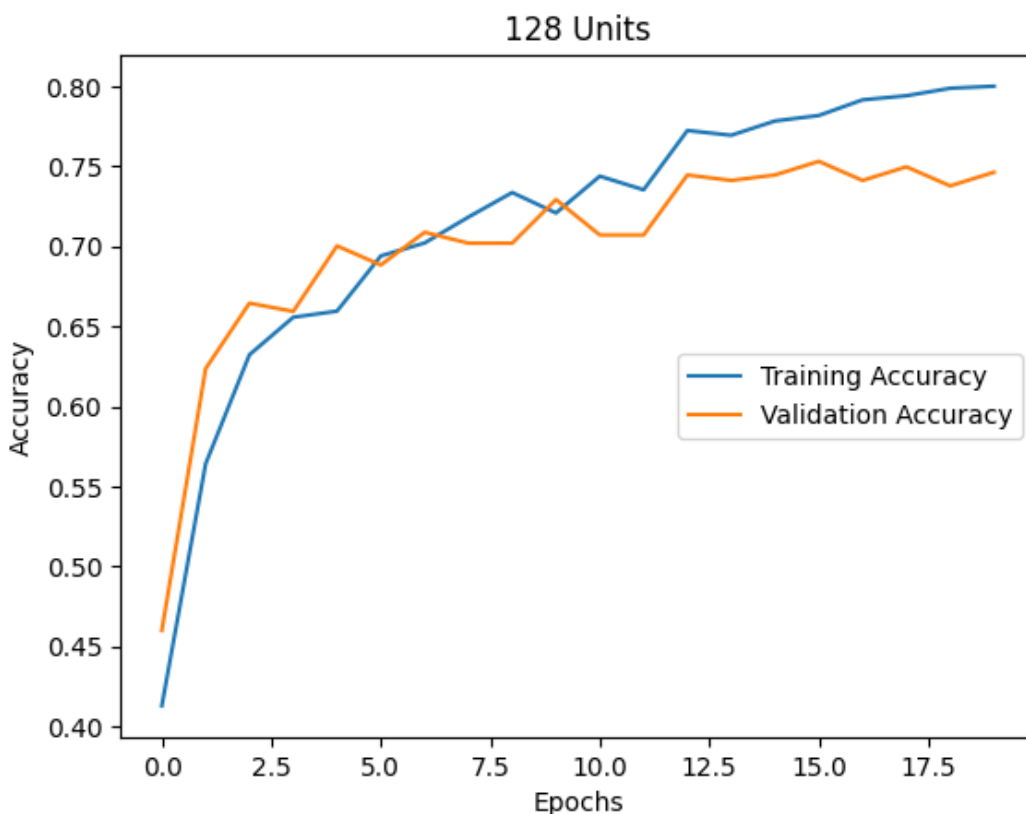
Epoch 11/20

74/74  34s 448ms/step - accuracy: 0.7568 - loss: 0.6222 - val_accuracy: 0.7070 - val_loss: 0.8022

```
y: 0.7070 - val_loss: 0.8033
Epoch 12/20
74/74 34s 445ms/step - accuracy: 0.7293 - loss: 0.6938 - val_accuracy: 0.7070 - val_loss: 0.7714
Epoch 13/20
74/74 34s 449ms/step - accuracy: 0.7644 - loss: 0.6260 - val_accuracy: 0.7445 - val_loss: 0.7194
Epoch 14/20
74/74 34s 448ms/step - accuracy: 0.7685 - loss: 0.5715 - val_accuracy: 0.7411 - val_loss: 0.6913
Epoch 15/20
74/74 35s 456ms/step - accuracy: 0.7943 - loss: 0.5612 - val_accuracy: 0.7445 - val_loss: 0.7016
Epoch 16/20
74/74 34s 452ms/step - accuracy: 0.7653 - loss: 0.5751 - val_accuracy: 0.7530 - val_loss: 0.6939
Epoch 17/20
74/74 34s 446ms/step - accuracy: 0.8019 - loss: 0.5120 - val_accuracy: 0.7411 - val_loss: 0.7295
Epoch 18/20
74/74 34s 447ms/step - accuracy: 0.7940 - loss: 0.5165 - val_accuracy: 0.7496 - val_loss: 0.6914
Epoch 19/20
74/74 34s 448ms/step - accuracy: 0.8059 - loss: 0.4945 - val_accuracy: 0.7376 - val_loss: 0.7416
Epoch 20/20
74/74 34s 450ms/step - accuracy: 0.8003 - loss: 0.5171 - val_accuracy: 0.7462 - val_loss: 0.7008
```

In [139]:

```
# Creating dataframe
history_df = pd.DataFrame(history01.history)
# Plotting
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("128 Units")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("128 Units")
plt.show()
```



FC Layer with 256 units (Best Model with Highest Validation Accuracy)

In [33]:

```
model02 = Sequential()
# First Convolutional and pooling layer
model02.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model02.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model02.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model02.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model02.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model02.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model02.add(Flatten())
# Dense layer with 256 neurons
model02.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model02.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model02.add(Dense(5, activation = 'softmax'))
```

In [34]:

```
model02.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history02 = model02.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))
```

Epoch 1/20

74/74 ————— 42s 517ms/step - accuracy: 0.3455 - loss: 1.5171 - val_accuracy: 0.5537 - val_loss: 1.1185

Epoch 2/20

74/74 ————— 35s 460ms/step - accuracy: 0.5475 - loss: 1.1442 - val_accuracy: 0.6508 - val_loss: 0.9726

Epoch 3/20

74/74 ————— 35s 459ms/step - accuracy: 0.6117 - loss: 0.9974 - val_accuracy: 0.6474 - val_loss: 0.9552

Epoch 4/20

74/74 ————— 35s 463ms/step - accuracy: 0.6267 - loss: 0.9175 - val_accuracy: 0.6644 - val_loss: 0.8767

Epoch 5/20

74/74 ————— 35s 461ms/step - accuracy: 0.6537 - loss: 0.8724 - val_accuracy: 0.6831 - val_loss: 0.8432

Epoch 6/20

74/74 ————— 37s 488ms/step - accuracy: 0.6616 - loss: 0.8438 - val_accuracy: 0.6780 - val_loss: 0.8634

Epoch 7/20

74/74 ————— 35s 461ms/step - accuracy: 0.7099 - loss: 0.7773 - val_accuracy: 0.6882 - val_loss: 0.8343

Epoch 8/20

74/74 ————— 36s 470ms/step - accuracy: 0.7278 - loss: 0.7174 - val_accuracy: 0.7172 - val_loss: 0.7601

Epoch 9/20

74/74 ————— 35s 464ms/step - accuracy: 0.7240 - loss: 0.7060 - val_accuracy: 0.7325 - val_loss: 0.7400

Epoch 10/20

74/74 ————— 35s 463ms/step - accuracy: 0.7336 - loss: 0.6777 - val_accuracy: 0.7257 - val_loss: 0.7540

Epoch 11/20

74/74 ————— 37s 494ms/step - accuracy: 0.7485 - loss: 0.6531 - val_accuracy: 0.7070 - val_loss: 0.8946

```

Epoch 12/20
74/74 35s 463ms/step - accuracy: 0.7439 - loss: 0.6780 - val_accu
racy: 0.7342 - val_loss: 0.7615
Epoch 13/20
74/74 35s 465ms/step - accuracy: 0.7722 - loss: 0.6150 - val_accu
racy: 0.7359 - val_loss: 0.7531
Epoch 14/20
74/74 35s 464ms/step - accuracy: 0.7697 - loss: 0.6123 - val_accu
racy: 0.7411 - val_loss: 0.7010
Epoch 15/20
74/74 35s 465ms/step - accuracy: 0.7680 - loss: 0.5749 - val_accu
racy: 0.7445 - val_loss: 0.6706
Epoch 16/20
74/74 35s 466ms/step - accuracy: 0.8058 - loss: 0.5168 - val_accu
racy: 0.7445 - val_loss: 0.7166
Epoch 17/20
74/74 35s 464ms/step - accuracy: 0.7929 - loss: 0.5686 - val_accu
racy: 0.7547 - val_loss: 0.6795
Epoch 18/20
74/74 37s 484ms/step - accuracy: 0.8112 - loss: 0.5032 - val_accu
racy: 0.7530 - val_loss: 0.7015
Epoch 19/20
74/74 37s 493ms/step - accuracy: 0.8093 - loss: 0.4945 - val_accu
racy: 0.7547 - val_loss: 0.7309
Epoch 20/20
74/74 35s 464ms/step - accuracy: 0.8144 - loss: 0.4920 - val_accu
racy: 0.7888 - val_loss: 0.6835

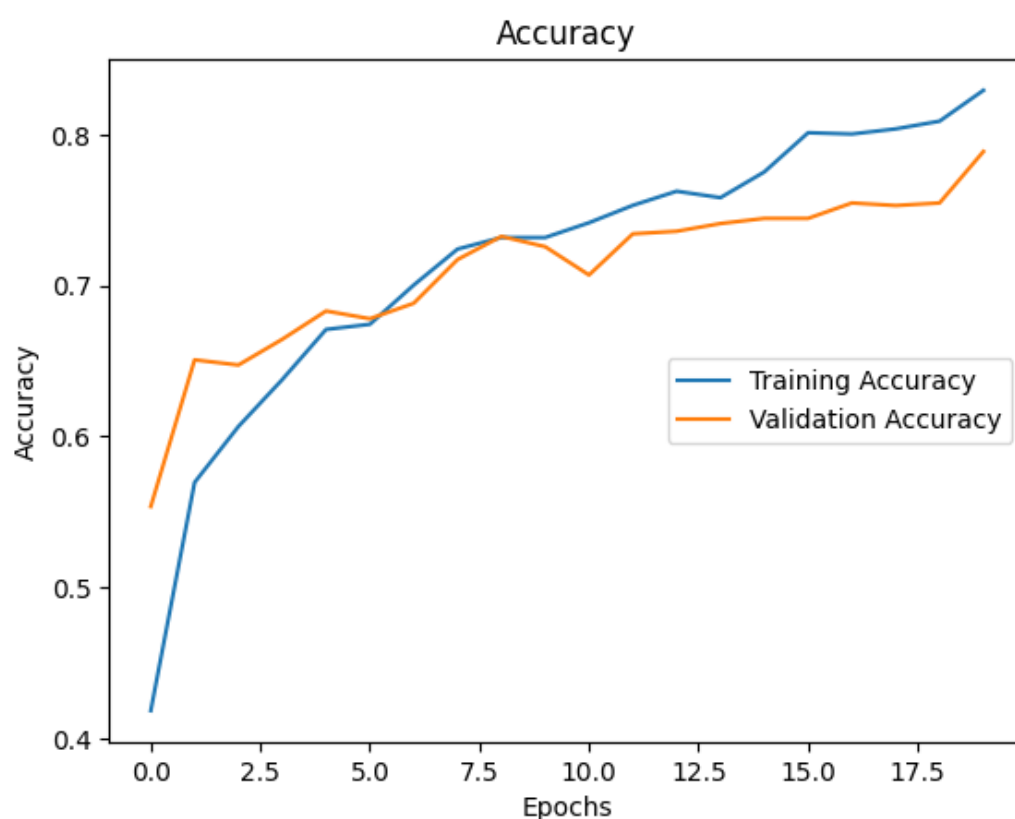
```

In [141]:

```

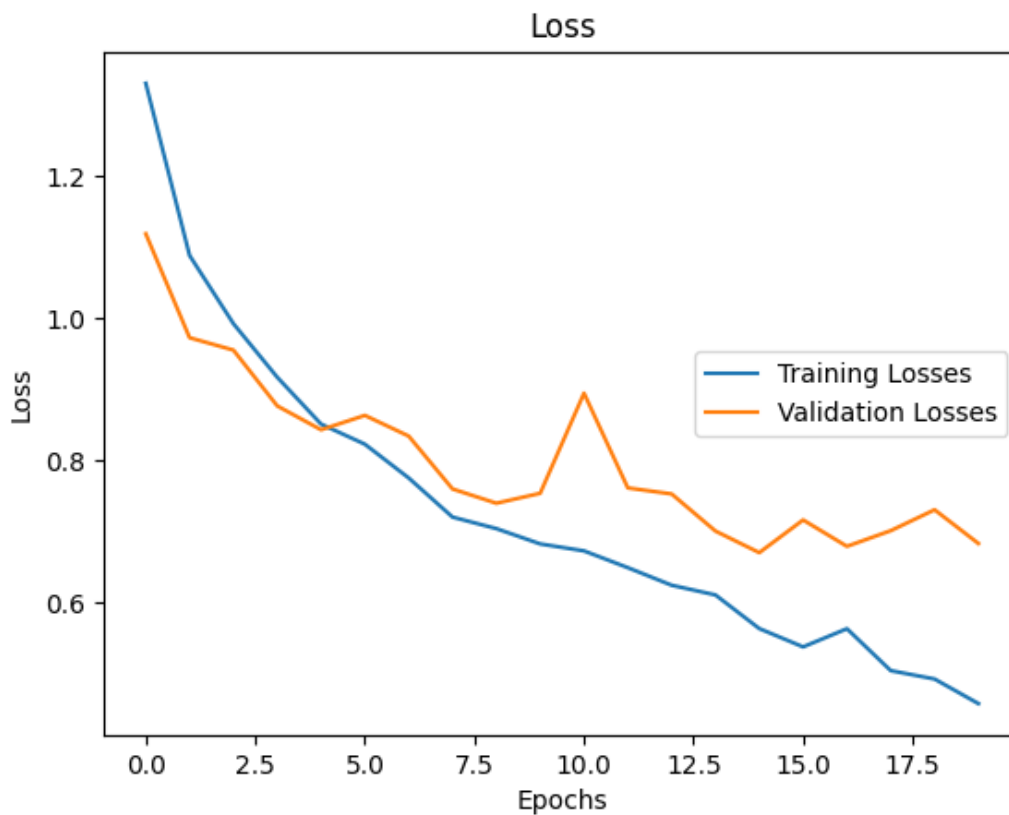
# Creating dataframe
history_df = pd.DataFrame(history02.history)
# Plotting accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("256 Units Accuracy")
plt.show()

```



In [142]:


```
# Plotting Loss
plt.plot(history_df["loss"], label = "Training Losses")
plt.plot(history_df["val_loss"], label = "Validation Losses")
plt.legend(loc=7)
plt.title("Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.savefig("256 Units Loss")
plt.show()
```



FC layers with 512 units

In [143]:

```
model = Sequential()
# First Convolutional and pooling layer
model.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model.add(Flatten())
# Dense layer with 512 neurons
model.add(Dense(512, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model.add(Dense(5, activation = 'softmax'))
# Summary
model.summary()
```

C:\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_34"

Layer (type)	Output Shape	Param #
conv2d_102 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_102 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_103 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_103 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_104 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_104 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_34 (Flatten)	(None, 25088)	0
dense_68 (Dense)	(None, 512)	12,845,568
dropout_28 (Dropout)	(None, 512)	0
dense_69 (Dense)	(None, 5)	2,565

Total params: 12,941,381 (49.37 MB)

Trainable params: 12,941,381 (49.37 MB)

Non-trainable params: 0 (0.00 B)

In [36]:

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

batch_size = 32
history = model.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 20, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation"))
```

Epoch 1/20

74/74 ————— 46s 586ms/step - accuracy: 0.2846 - loss: 1.6835 - val_accuracy: 0.5043 - val_loss: 1.2235

Epoch 2/20

74/74 ————— 41s 543ms/step - accuracy: 0.5136 - loss: 1.1843 - val_accuracy: 0.5997 - val_loss: 1.0105

Epoch 3/20

74/74 ————— 38s 499ms/step - accuracy: 0.6115 - loss: 0.9652 - val_accuracy: 0.6235 - val_loss: 0.9615

Epoch 4/20

74/74 ————— 39s 520ms/step - accuracy: 0.6224 - loss: 0.9465 - val_accuracy: 0.6644 - val_loss: 0.8792

Epoch 5/20

74/74 ————— 38s 499ms/step - accuracy: 0.6751 - loss: 0.8450 - val_accuracy: 0.6797 - val_loss: 0.8341

Epoch 6/20

74/74 ————— 38s 500ms/step - accuracy: 0.6816 - loss: 0.8056 - val_accuracy: 0.7019 - val_loss: 0.8259

Epoch 7/20

74/74 ————— 38s 504ms/step - accuracy: 0.6905 - loss: 0.7857 - val_accuracy: 0.7155 - val_loss: 0.7866

Epoch 8/20

74/74 ————— 38s 502ms/step - accuracy: 0.7192 - loss: 0.7207 - val_accuracy: 0.7155 - val_loss: 0.7813

Epoch 9/20

74/74 ————— 38s 500ms/step - accuracy: 0.7293 - loss: 0.7133 - val accuracy: 0.7293 - val_loss: 0.7133

```

y: 0.7206 - val_loss: 0.7681
Epoch 10/20
74/74 ██████████ 38s 502ms/step - accuracy: 0.7550 - loss: 0.6896 - val_accu
y: 0.7189 - val_loss: 0.7762
Epoch 11/20
74/74 ██████████ 38s 508ms/step - accuracy: 0.7749 - loss: 0.6237 - val_accu
y: 0.7411 - val_loss: 0.7068
Epoch 12/20
74/74 ██████████ 39s 513ms/step - accuracy: 0.7613 - loss: 0.6049 - val_accu
y: 0.7257 - val_loss: 0.7164
Epoch 13/20
74/74 ██████████ 38s 500ms/step - accuracy: 0.7890 - loss: 0.5692 - val_accu
y: 0.7189 - val_loss: 0.7380
Epoch 14/20
74/74 ██████████ 40s 520ms/step - accuracy: 0.7657 - loss: 0.5880 - val_accu
y: 0.7479 - val_loss: 0.6872
Epoch 15/20
74/74 ██████████ 38s 505ms/step - accuracy: 0.7888 - loss: 0.5334 - val_accu
y: 0.7496 - val_loss: 0.7311
Epoch 16/20
74/74 ██████████ 38s 503ms/step - accuracy: 0.8167 - loss: 0.5060 - val_accu
y: 0.7479 - val_loss: 0.7235
Epoch 17/20
74/74 ██████████ 38s 504ms/step - accuracy: 0.8151 - loss: 0.4755 - val_accu
y: 0.7155 - val_loss: 0.7860
Epoch 18/20
74/74 ██████████ 38s 503ms/step - accuracy: 0.7813 - loss: 0.5598 - val_accu
y: 0.7700 - val_loss: 0.7276
Epoch 19/20
74/74 ██████████ 38s 504ms/step - accuracy: 0.8185 - loss: 0.4927 - val_accu
y: 0.7445 - val_loss: 0.7212
Epoch 20/20
74/74 ██████████ 39s 513ms/step - accuracy: 0.8377 - loss: 0.4245 - val_accu
y: 0.7836 - val_loss: 0.7239

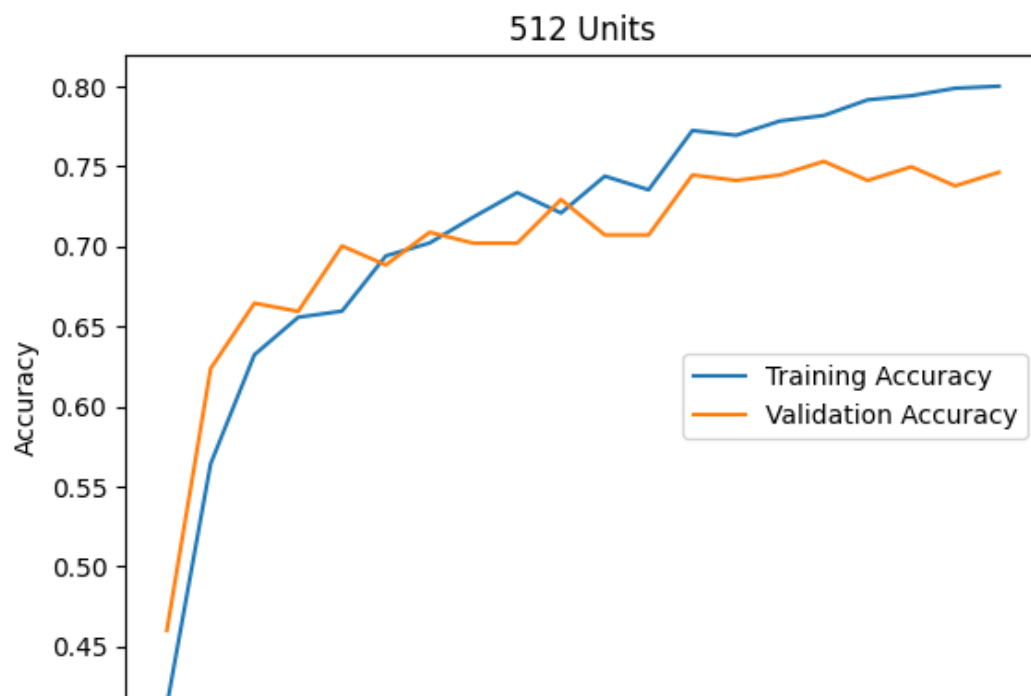
```

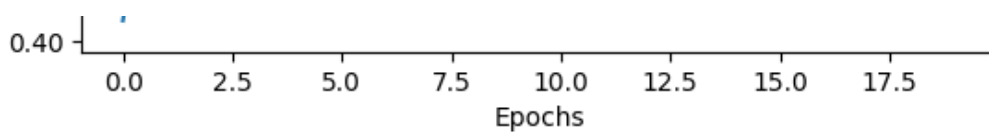
In [144]:

```

# Creating dataframe
history_df = pd.DataFrame(history01.history)
# Plotting accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("512 Units")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("512 Units")
plt.show()

```





No dropout

In [85]:

```
modeln = Sequential()
# First Convolutional and pooling layer
modeln.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
modeln.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
modeln.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
modeln.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
modeln.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
modeln.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
modeln.add(Flatten())
# Dense layer with 64 neurons
modeln.add(Dense(64, activation = 'relu'))
# Output dense layer with 5 neuron and softmax act function
modeln.add(Dense(5, activation = 'softmax'))
```

In [86]:

```
modeln.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
historyn = modeln.fit(train_datagen.flow(x_train, y_train_cat,
                                          batch_size = batch_size,
                                          subset = "training"),
                      epochs = 20, validation_data =
                      train_datagen.flow(x_train, y_train_cat,
                                          batch_size = batch_size,
                                          subset = "validation"))
```

Epoch 1/20

74/74 ————— 40s 495ms/step - accuracy: 0.2827 - loss: 1.6431 - val_accuracy: 0.4736 - val_loss: 1.2588

Epoch 2/20

74/74 ————— 36s 469ms/step - accuracy: 0.4958 - loss: 1.1974 - val_accuracy: 0.5588 - val_loss: 1.1268

Epoch 3/20

74/74 ————— 38s 498ms/step - accuracy: 0.5188 - loss: 1.1154 - val_accuracy: 0.5605 - val_loss: 1.1109

Epoch 4/20

74/74 ————— 35s 453ms/step - accuracy: 0.5723 - loss: 1.0540 - val_accuracy: 0.6031 - val_loss: 1.0312

Epoch 5/20

74/74 ————— 33s 433ms/step - accuracy: 0.6255 - loss: 0.9546 - val_accuracy: 0.6525 - val_loss: 0.9641

Epoch 6/20

74/74 ————— 33s 437ms/step - accuracy: 0.6652 - loss: 0.8588 - val_accuracy: 0.6763 - val_loss: 0.8882

Epoch 7/20

74/74 ————— 37s 486ms/step - accuracy: 0.6694 - loss: 0.8188 - val_accuracy: 0.6848 - val_loss: 0.8796

Epoch 8/20

74/74 ————— 33s 440ms/step - accuracy: 0.6897 - loss: 0.7792 - val_accuracy: 0.6934 - val_loss: 0.8401

Epoch 9/20

74/74 ————— 33s 440ms/step - accuracy: 0.7047 - loss: 0.7639 - val_accuracy: 0.6934 - val_loss: 0.8165

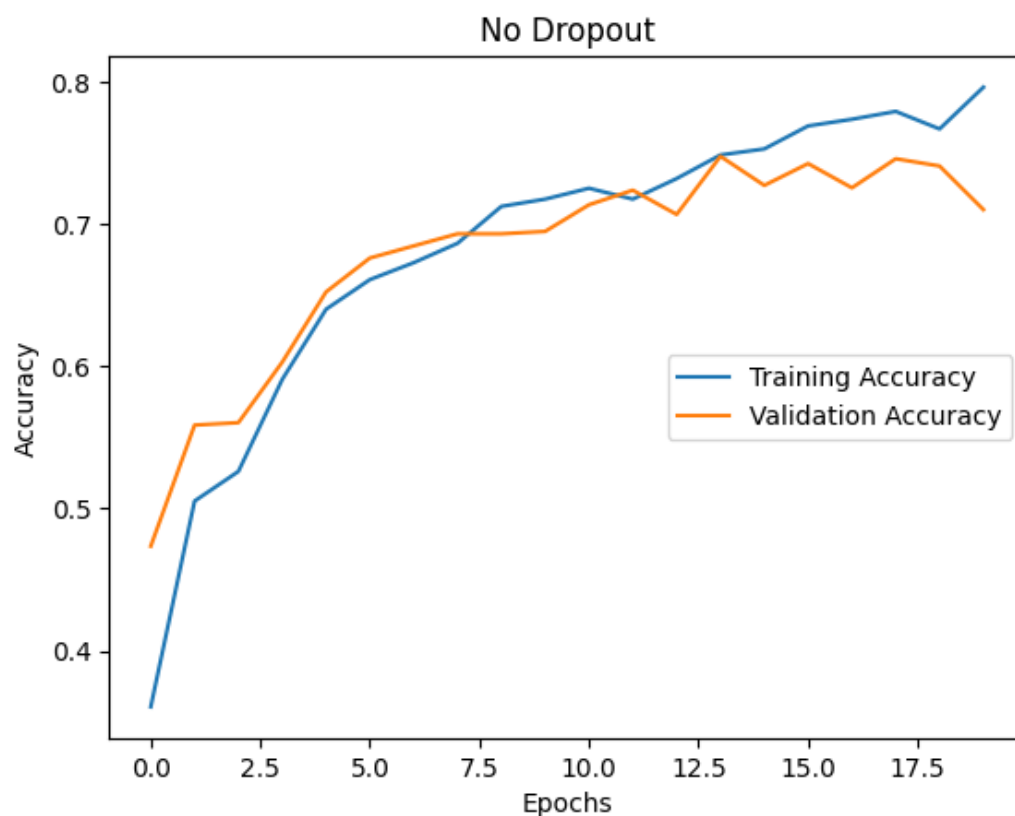
Epoch 10/20

74/74 ————— 33s 439ms/step - accuracy: 0.7248 - loss: 0.7139 - val_accuracy: 0.6951 - val loss: 0.8440

Epoch 11/20
74/74 33s 436ms/step - accuracy: 0.7317 - loss: 0.6879 - val_accuracy: 0.7138 - val_loss: 0.8023
Epoch 12/20
74/74 33s 438ms/step - accuracy: 0.7246 - loss: 0.7247 - val_accuracy: 0.7240 - val_loss: 0.7849
Epoch 13/20
74/74 34s 441ms/step - accuracy: 0.7268 - loss: 0.7088 - val_accuracy: 0.7070 - val_loss: 0.7710
Epoch 14/20
74/74 33s 435ms/step - accuracy: 0.7520 - loss: 0.6738 - val_accuracy: 0.7479 - val_loss: 0.7539
Epoch 15/20
74/74 34s 450ms/step - accuracy: 0.7462 - loss: 0.6260 - val_accuracy: 0.7274 - val_loss: 0.7865
Epoch 16/20
74/74 33s 438ms/step - accuracy: 0.7797 - loss: 0.5662 - val_accuracy: 0.7428 - val_loss: 0.7333
Epoch 17/20
74/74 34s 452ms/step - accuracy: 0.7697 - loss: 0.6004 - val_accuracy: 0.7257 - val_loss: 0.7780
Epoch 18/20
74/74 36s 477ms/step - accuracy: 0.7665 - loss: 0.5783 - val_accuracy: 0.7462 - val_loss: 0.7311
Epoch 19/20
74/74 33s 438ms/step - accuracy: 0.7831 - loss: 0.5672 - val_accuracy: 0.7411 - val_loss: 0.7550
Epoch 20/20
74/74 34s 442ms/step - accuracy: 0.8034 - loss: 0.5142 - val_accuracy: 0.7104 - val_loss: 0.8256

In [145]:

```
# Creating Dataframe
history_df = pd.DataFrame(historyn.history)
#Plotting Accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("No Dropout")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("No Dropout")
plt.show()
```



Learning rate of 0.0001

In [43]:

```
model21 = Sequential()
# First Convolutional and pooling layer
model21.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model21.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model21.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model21.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model21.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model21.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model21.add(Flatten())
# Dense layer with 256 neurons
model21.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model21.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model21.add(Dense(5, activation = 'softmax'))
```

In [44]:

```
adam_optimizer = Adam(learning_rate = 0.0001)

model21.compile(optimizer=adam_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history21 = model21.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))
```

Epoch 1/20

74/74 ————— 39s 492ms/step - accuracy: 0.2848 - loss: 1.5324 - val_accuracy: 0.4889 - val_loss: 1.2273

Epoch 2/20

74/74 ————— 36s 469ms/step - accuracy: 0.4529 - loss: 1.2488 - val_accuracy: 0.5400 - val_loss: 1.1324

Epoch 3/20

74/74 ————— 36s 472ms/step - accuracy: 0.5085 - loss: 1.1375 - val_accuracy: 0.5758 - val_loss: 1.1049

Epoch 4/20

74/74 ————— 37s 486ms/step - accuracy: 0.5494 - loss: 1.0956 - val_accuracy: 0.5997 - val_loss: 1.0529

Epoch 5/20

74/74 ————— 36s 479ms/step - accuracy: 0.5687 - loss: 1.0715 - val_accuracy: 0.6422 - val_loss: 1.0044

Epoch 6/20

74/74 ————— 35s 464ms/step - accuracy: 0.6207 - loss: 1.0090 - val_accuracy: 0.6491 - val_loss: 0.9916

Epoch 7/20

74/74 ————— 35s 466ms/step - accuracy: 0.6170 - loss: 0.9682 - val_accuracy: 0.6729 - val_loss: 0.9479

Epoch 8/20

74/74 ————— 35s 465ms/step - accuracy: 0.6303 - loss: 0.9241 - val_accuracy: 0.6542 - val_loss: 0.9774

Epoch 9/20

74/74 ————— 35s 464ms/step - accuracy: 0.6359 - loss: 0.9481 - val_accuracy: 0.6746 - val_loss: 0.9112

Epoch 10/20

74/74 ————— 35s 461ms/step - accuracy: 0.6487 - loss: 0.9036 - val_accuracy:

```

y: 0.6934 - val_loss: 0.8772
Epoch 11/20
74/74 ————— 36s 470ms/step - accuracy: 0.6770 - loss: 0.8736 - val_accu
y: 0.6968 - val_loss: 0.8326
Epoch 12/20
74/74 ————— 35s 465ms/step - accuracy: 0.6912 - loss: 0.8205 - val_accu
y: 0.6848 - val_loss: 0.8828
Epoch 13/20
74/74 ————— 35s 462ms/step - accuracy: 0.6928 - loss: 0.8032 - val_accu
y: 0.6934 - val_loss: 0.8380
Epoch 14/20
74/74 ————— 35s 463ms/step - accuracy: 0.6729 - loss: 0.8289 - val_accu
y: 0.6934 - val_loss: 0.8097
Epoch 15/20
74/74 ————— 35s 464ms/step - accuracy: 0.6838 - loss: 0.7890 - val_accu
y: 0.6968 - val_loss: 0.8203
Epoch 16/20
74/74 ————— 35s 463ms/step - accuracy: 0.6940 - loss: 0.7968 - val_accu
y: 0.6917 - val_loss: 0.8307
Epoch 17/20
74/74 ————— 35s 466ms/step - accuracy: 0.7042 - loss: 0.7824 - val_accu
y: 0.7138 - val_loss: 0.8365
Epoch 18/20
74/74 ————— 35s 464ms/step - accuracy: 0.7085 - loss: 0.7479 - val_accu
y: 0.7172 - val_loss: 0.7825
Epoch 19/20
74/74 ————— 36s 467ms/step - accuracy: 0.7259 - loss: 0.7383 - val_accu
y: 0.7053 - val_loss: 0.8146
Epoch 20/20
74/74 ————— 35s 464ms/step - accuracy: 0.7239 - loss: 0.7288 - val_accu
y: 0.7138 - val_loss: 0.7787

```

In [146]:

```

# Creating dataframe
history_df = pd.DataFrame(history21.history)
# Plotting
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("Learning Rate = 0.0001")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("Learning rate of 1e-3")
plt.show()

```



Batch size of 64


In [46]:

```
model31 = Sequential()
# First Convolutional and pooling layer
model31.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model31.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model31.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model31.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model31.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model31.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model31.add(Flatten())
# Dense layer with 256 neurons
model31.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model31.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model31.add(Dense(5, activation = 'softmax'))


#Compilation
model31.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 64
history31 = model31.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))
```


Epoch 1/20

37/37  36s 883ms/step - accuracy: 0.2692 - loss: 1.7685 - val_accuracy: 0.4804 - val_loss: 1.2715


Epoch 2/20

37/37  34s 866ms/step - accuracy: 0.4906 - loss: 1.2218 - val_accuracy: 0.5826 - val_loss: 1.1120


Epoch 3/20

37/37  35s 909ms/step - accuracy: 0.5587 - loss: 1.0981 - val_accuracy: 0.5826 - val_loss: 1.0865


Epoch 4/20

37/37  34s 874ms/step - accuracy: 0.5913 - loss: 1.0432 - val_accuracy: 0.6099 - val_loss: 1.0038


Epoch 5/20

37/37  36s 932ms/step - accuracy: 0.6098 - loss: 0.9701 - val_accuracy: 0.6763 - val_loss: 0.9161


Epoch 6/20

37/37  39s 1s/step - accuracy: 0.6453 - loss: 0.9169 - val_accuracy: 0.6627 - val_loss: 0.9438


Epoch 7/20

37/37  39s 1s/step - accuracy: 0.6398 - loss: 0.8692 - val_accuracy: 0.6848 - val_loss: 0.8829


Epoch 8/20

37/37  43s 1s/step - accuracy: 0.6854 - loss: 0.8215 - val_accuracy: 0.6559 - val_loss: 0.9149


Epoch 9/20

37/37  37s 948ms/step - accuracy: 0.6842 - loss: 0.8188 - val_accuracy: 0.6814 - val_loss: 0.8671

Epoch 10/20

37/37  37s 952ms/step - accuracy: 0.7286 - loss: 0.7457 - val_accuracy: 0.6814 - val_loss: 0.8288

Epoch 11/20

37/37  38s 977ms/step - accuracy: 0.7047 - loss: 0.7557 - val_accuracy: 0.7055 - val_loss: 0.7700


```

y: 0.7257 - val_loss: 0.7798
Epoch 12/20
37/37 ————— 39s 1s/step - accuracy: 0.7300 - loss: 0.6842 - val_accuracy:
0.6559 - val_loss: 0.9208
Epoch 13/20
37/37 ————— 37s 959ms/step - accuracy: 0.7203 - loss: 0.7131 - val_accu-
racy: 0.6695 - val_loss: 0.8550
Epoch 14/20
37/37 ————— 37s 957ms/step - accuracy: 0.7449 - loss: 0.6894 - val_accu-
racy: 0.7257 - val_loss: 0.7612
Epoch 15/20
37/37 ————— 37s 931ms/step - accuracy: 0.7589 - loss: 0.6417 - val_accu-
racy: 0.7104 - val_loss: 0.8128
Epoch 16/20
37/37 ————— 37s 941ms/step - accuracy: 0.7804 - loss: 0.6023 - val_accu-
racy: 0.7155 - val_loss: 0.7554
Epoch 17/20
37/37 ————— 38s 967ms/step - accuracy: 0.7697 - loss: 0.5942 - val_accu-
racy: 0.7496 - val_loss: 0.7037
Epoch 18/20
37/37 ————— 40s 1s/step - accuracy: 0.7728 - loss: 0.6000 - val_accuracy:
0.7394 - val_loss: 0.7343
Epoch 19/20
37/37 ————— 35s 888ms/step - accuracy: 0.7953 - loss: 0.5330 - val_accu-
racy: 0.7359 - val_loss: 0.7457
Epoch 20/20
37/37 ————— 34s 877ms/step - accuracy: 0.8054 - loss: 0.5221 - val_accu-
racy: 0.7513 - val_loss: 0.7324

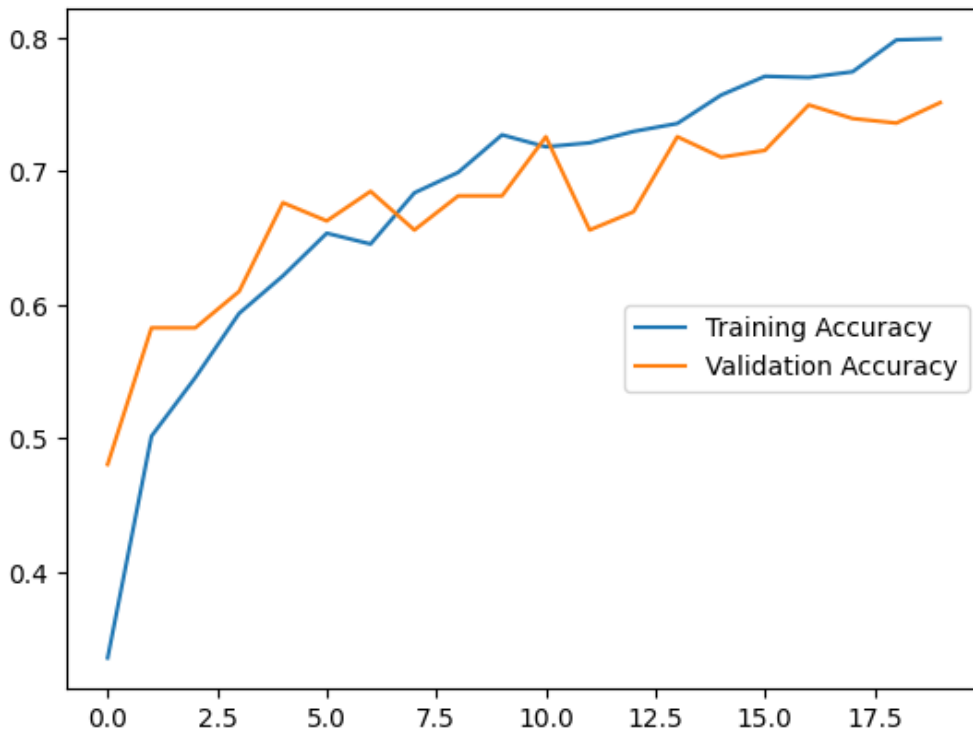
```

In [147]:

```

# Creating dataframe
history_df = pd.DataFrame(history31.history)
# Plotting Accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.savefig("64 Batch Size")
plt.show()

```



128 Batch

In [48]:

```
model311 = Sequential()
```

```

# First Convolutional and pooling layer
model311.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model311.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model311.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model311.add(MaxPooling2D(pool_size = (2, 2)))
# Third Convolutional and pooling layer
model311.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model311.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model311.add(Flatten())
# Dense layer with 256 neurons
model311.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model311.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model311.add(Dense(5, activation = 'softmax'))

#Compilation
model311.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 128
history311 = model311.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
train_datagen.flow(x_train, y_train_cat,
                    batch_size = batch_size,
                    subset = "validation"))

```

```

Epoch 1/20
19/19 ————— 38s 2s/step - accuracy: 0.2283 - loss: 1.8896 - val_accuracy:
0.4668 - val_loss: 1.3691
Epoch 2/20
19/19 ————— 35s 2s/step - accuracy: 0.4261 - loss: 1.3259 - val_accuracy:
0.4770 - val_loss: 1.2388
Epoch 3/20
19/19 ————— 35s 2s/step - accuracy: 0.4898 - loss: 1.2133 - val_accuracy:
0.4702 - val_loss: 1.2853
Epoch 4/20
19/19 ————— 35s 2s/step - accuracy: 0.5074 - loss: 1.1937 - val_accuracy:
0.6031 - val_loss: 1.0387
Epoch 5/20
19/19 ————— 38s 2s/step - accuracy: 0.5654 - loss: 1.0634 - val_accuracy:
0.6048 - val_loss: 1.0333
Epoch 6/20
19/19 ————— 35s 2s/step - accuracy: 0.6246 - loss: 0.9806 - val_accuracy:
0.6474 - val_loss: 0.9736
Epoch 7/20
19/19 ————— 35s 2s/step - accuracy: 0.6396 - loss: 0.9402 - val_accuracy:
0.6729 - val_loss: 0.9177
Epoch 8/20
19/19 ————— 35s 2s/step - accuracy: 0.6512 - loss: 0.9092 - val_accuracy:
0.6388 - val_loss: 0.9924
Epoch 9/20
19/19 ————— 35s 2s/step - accuracy: 0.6524 - loss: 0.9063 - val_accuracy:
0.6882 - val_loss: 0.8790
Epoch 10/20
19/19 ————— 35s 2s/step - accuracy: 0.6899 - loss: 0.8381 - val_accuracy:
0.6797 - val_loss: 0.9024
Epoch 11/20
19/19 ————— 35s 2s/step - accuracy: 0.6853 - loss: 0.8259 - val_accuracy:
0.6712 - val_loss: 0.8901
Epoch 12/20
19/19 ————— 35s 2s/step - accuracy: 0.7203 - loss: 0.7627 - val_accuracy:
0.7223 - val_loss: 0.8416
Epoch 13/20
19/19 ————— 35s 2s/step - accuracy: 0.7132 - loss: 0.7567 - val_accuracy:
0.7053 - val_loss: 0.8027
Epoch 14/20

```

```

19/19 ————— 35s 2s/step - accuracy: 0.7359 - loss: 0.7091 - val_accuracy:
0.7087 - val_loss: 0.7930
Epoch 15/20
19/19 ————— 35s 2s/step - accuracy: 0.7456 - loss: 0.6621 - val_accuracy:
0.7223 - val_loss: 0.7760
Epoch 16/20
19/19 ————— 35s 2s/step - accuracy: 0.7288 - loss: 0.7097 - val_accuracy:
0.7019 - val_loss: 0.8235
Epoch 17/20
19/19 ————— 35s 2s/step - accuracy: 0.7337 - loss: 0.6841 - val_accuracy:
0.6899 - val_loss: 0.9017
Epoch 18/20
19/19 ————— 34s 2s/step - accuracy: 0.7346 - loss: 0.6774 - val_accuracy:
0.7240 - val_loss: 0.7428
Epoch 19/20
19/19 ————— 34s 2s/step - accuracy: 0.7563 - loss: 0.6404 - val_accuracy:
0.6934 - val_loss: 0.8260
Epoch 20/20
19/19 ————— 35s 2s/step - accuracy: 0.7425 - loss: 0.6633 - val_accuracy:
0.7308 - val_loss: 0.7368

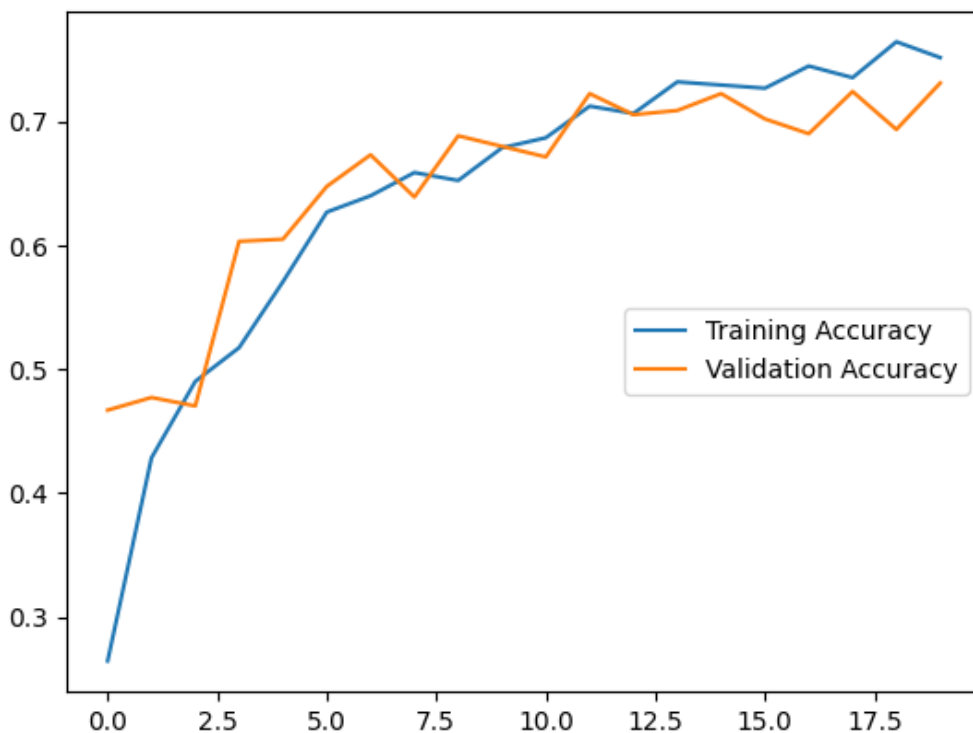
```

In [148]:

```

# Creating dataframe
history_df = pd.DataFrame(history311.history)
# Plotting Accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.savefig("Batch Size 128")
plt.show()

```



Kernel Size of 5 x 5

In [58]:

```

model41 = Sequential()
# First Convolutional and pooling layer
model41.add(Conv2D(filters = 32, kernel_size = (5, 5), input_shape = (128, 128, 3), activation = 'relu'))
model41.add(MaxPooling2D(pool_size = (2, 2)))
# Second Convolutional and pooling layer
model41.add(Conv2D(filters = 64, kernel_size = (5, 5), activation = 'relu'))
model41.add(MaxPooling2D(pool_size = (2, 2)))

```

```

# Third Convolutional and pooling layer
model41.add(Conv2D(filters = 128, kernel_size = (5, 5), activation = 'relu'))
model41.add(MaxPooling2D(pool_size = (2, 2)))
# Adding Flatten Layer
model41.add(Flatten())
# Dense layer with 256 neurons
model41.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model41.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model41.add(Dense(5, activation = 'softmax'))

# Compilation
model41.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history41 = model41.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))

```

```

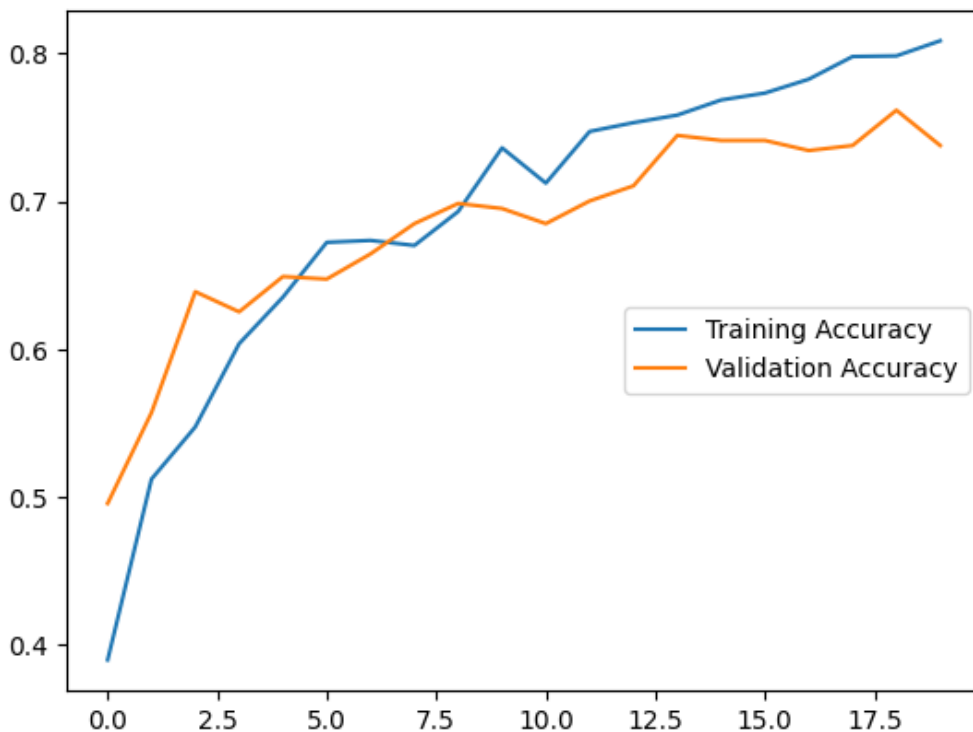
Epoch 1/20
74/74 ██████████ 61s 790ms/step - accuracy: 0.3248 - loss: 1.5170 - val_accu-
racy: 0.4957 - val_loss: 1.2234
Epoch 2/20
74/74 ██████████ 59s 779ms/step - accuracy: 0.5181 - loss: 1.1698 - val_accu-
racy: 0.5571 - val_loss: 1.1508
Epoch 3/20
74/74 ██████████ 66s 885ms/step - accuracy: 0.5522 - loss: 1.1155 - val_accu-
racy: 0.6388 - val_loss: 1.0092
Epoch 4/20
74/74 ██████████ 65s 866ms/step - accuracy: 0.6078 - loss: 1.0460 - val_accu-
racy: 0.6252 - val_loss: 1.0042
Epoch 5/20
74/74 ██████████ 65s 864ms/step - accuracy: 0.6170 - loss: 0.9635 - val_accu-
racy: 0.6491 - val_loss: 0.9449
Epoch 6/20
74/74 ██████████ 63s 831ms/step - accuracy: 0.6737 - loss: 0.8601 - val_accu-
racy: 0.6474 - val_loss: 0.9082
Epoch 7/20
74/74 ██████████ 65s 867ms/step - accuracy: 0.6642 - loss: 0.8730 - val_accu-
racy: 0.6644 - val_loss: 0.8795
Epoch 8/20
74/74 ██████████ 63s 839ms/step - accuracy: 0.6574 - loss: 0.8486 - val_accu-
racy: 0.6848 - val_loss: 0.8808
Epoch 9/20
74/74 ██████████ 62s 823ms/step - accuracy: 0.6876 - loss: 0.8088 - val_accu-
racy: 0.6985 - val_loss: 0.8187
Epoch 10/20
74/74 ██████████ 65s 862ms/step - accuracy: 0.7530 - loss: 0.7086 - val_accu-
racy: 0.6951 - val_loss: 0.8602
Epoch 11/20
74/74 ██████████ 64s 857ms/step - accuracy: 0.6982 - loss: 0.7408 - val_accu-
racy: 0.6848 - val_loss: 0.8509
Epoch 12/20
74/74 ██████████ 61s 810ms/step - accuracy: 0.7399 - loss: 0.6962 - val_accu-
racy: 0.7002 - val_loss: 0.8286
Epoch 13/20
74/74 ██████████ 64s 853ms/step - accuracy: 0.7647 - loss: 0.6368 - val_accu-
racy: 0.7104 - val_loss: 0.8145
Epoch 14/20
74/74 ██████████ 67s 897ms/step - accuracy: 0.7513 - loss: 0.6495 - val_accu-
racy: 0.7445 - val_loss: 0.7208
Epoch 15/20
74/74 ██████████ 63s 846ms/step - accuracy: 0.7702 - loss: 0.5836 - val_accu-
racy: 0.7411 - val_loss: 0.7944
Epoch 16/20
74/74 ██████████ 59s 785ms/step - accuracy: 0.7738 - loss: 0.5961 - val_accu-
racy: 0.7411 - val_loss: 0.7773

```

Epoch 17/20
74/74 ————— 59s 787ms/step - accuracy: 0.7849 - loss: 0.5764 - val_accuracy: 0.7342 - val_loss: 0.8426
Epoch 18/20
74/74 ————— 60s 793ms/step - accuracy: 0.7927 - loss: 0.5327 - val_accuracy: 0.7376 - val_loss: 0.7503
Epoch 19/20
74/74 ————— 60s 801ms/step - accuracy: 0.8067 - loss: 0.5264 - val_accuracy: 0.7615 - val_loss: 0.6754
Epoch 20/20
74/74 ————— 62s 829ms/step - accuracy: 0.8005 - loss: 0.5143 - val_accuracy: 0.7376 - val_loss: 0.7430

In [149]:

```
# Creating dataframe
history_df = pd.DataFrame(history41.history)
# Plotting Accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.savefig("Kernel 5 x 5")
plt.show()
```



Batch Normalization

In [119]:

```
from tensorflow.keras.layers import BatchNormalization
model51 = Sequential()
# First Convolutional and pooling layer
model51.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model51.add(MaxPooling2D(pool_size = (2, 2)))
model51.add(BatchNormalization(momentum = 0.9))
# Second Convolutional and pooling layer
model51.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model51.add(MaxPooling2D(pool_size = (2, 2)))
model51.add(BatchNormalization(momentum = 0.9))
# Third Convolutional and pooling layer
model51.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model51.add(MaxPooling2D(pool_size = (2, 2)))
model51.add(BatchNormalization(momentum = 0.9))
# Adding Flatten Layer
```

```

model51.add(Flatten())
# Dense layer with 256 neurons
model51.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model51.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model51.add(Dense(5, activation = 'softmax'))

model51.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history51 = model51.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))

```

```

Epoch 1/20
74/74 ██████████ 46s 562ms/step - accuracy: 0.4124 - loss: 6.7480 - val_accuracy: 0.5230 - val_loss: 3.5849
Epoch 2/20
74/74 ██████████ 42s 558ms/step - accuracy: 0.4892 - loss: 3.7233 - val_accuracy: 0.4872 - val_loss: 3.1636
Epoch 3/20
74/74 ██████████ 45s 593ms/step - accuracy: 0.5037 - loss: 2.5391 - val_accuracy: 0.5605 - val_loss: 2.0459
Epoch 4/20
74/74 ██████████ 41s 533ms/step - accuracy: 0.5793 - loss: 1.7125 - val_accuracy: 0.6525 - val_loss: 1.4793
Epoch 5/20
74/74 ██████████ 44s 580ms/step - accuracy: 0.6239 - loss: 1.3857 - val_accuracy: 0.6610 - val_loss: 1.1782
Epoch 6/20
74/74 ██████████ 41s 543ms/step - accuracy: 0.6544 - loss: 1.0328 - val_accuracy: 0.6491 - val_loss: 1.3949
Epoch 7/20
74/74 ██████████ 40s 529ms/step - accuracy: 0.6743 - loss: 0.9299 - val_accuracy: 0.6610 - val_loss: 1.1834
Epoch 8/20
74/74 ██████████ 40s 527ms/step - accuracy: 0.6842 - loss: 0.8895 - val_accuracy: 0.7138 - val_loss: 0.9774
Epoch 9/20
74/74 ██████████ 40s 530ms/step - accuracy: 0.7005 - loss: 0.8068 - val_accuracy: 0.7053 - val_loss: 1.0025
Epoch 10/20
74/74 ██████████ 40s 530ms/step - accuracy: 0.7223 - loss: 0.7572 - val_accuracy: 0.7274 - val_loss: 0.7911
Epoch 11/20
74/74 ██████████ 40s 531ms/step - accuracy: 0.7405 - loss: 0.7184 - val_accuracy: 0.7172 - val_loss: 0.9004
Epoch 12/20
74/74 ██████████ 40s 535ms/step - accuracy: 0.7311 - loss: 0.7316 - val_accuracy: 0.7002 - val_loss: 0.9483
Epoch 13/20
74/74 ██████████ 40s 528ms/step - accuracy: 0.7341 - loss: 0.6922 - val_accuracy: 0.7155 - val_loss: 0.9361
Epoch 14/20
74/74 ██████████ 40s 530ms/step - accuracy: 0.7376 - loss: 0.6609 - val_accuracy: 0.7223 - val_loss: 0.9578
Epoch 15/20
74/74 ██████████ 40s 527ms/step - accuracy: 0.7684 - loss: 0.6765 - val_accuracy: 0.6899 - val_loss: 0.9025
Epoch 16/20
74/74 ██████████ 42s 556ms/step - accuracy: 0.7386 - loss: 0.7180 - val_accuracy: 0.7359 - val_loss: 0.8024
Epoch 17/20
74/74 ██████████ 43s 573ms/step - accuracy: 0.7783 - loss: 0.6059 - val_accuracy: 0.7172 - val_loss: 0.8960
Epoch 18/20
74/74 ██████████ 42s 549ms/step - accuracy: 0.7859 - loss: 0.5854 - val_accuracy:

```

y: 0.7308 - val_loss: 0.8384

Epoch 19/20

74/74 ————— 44s 583ms/step - accuracy: 0.7939 - loss: 0.5647 - val_accu

y: 0.7308 - val_loss: 0.8750

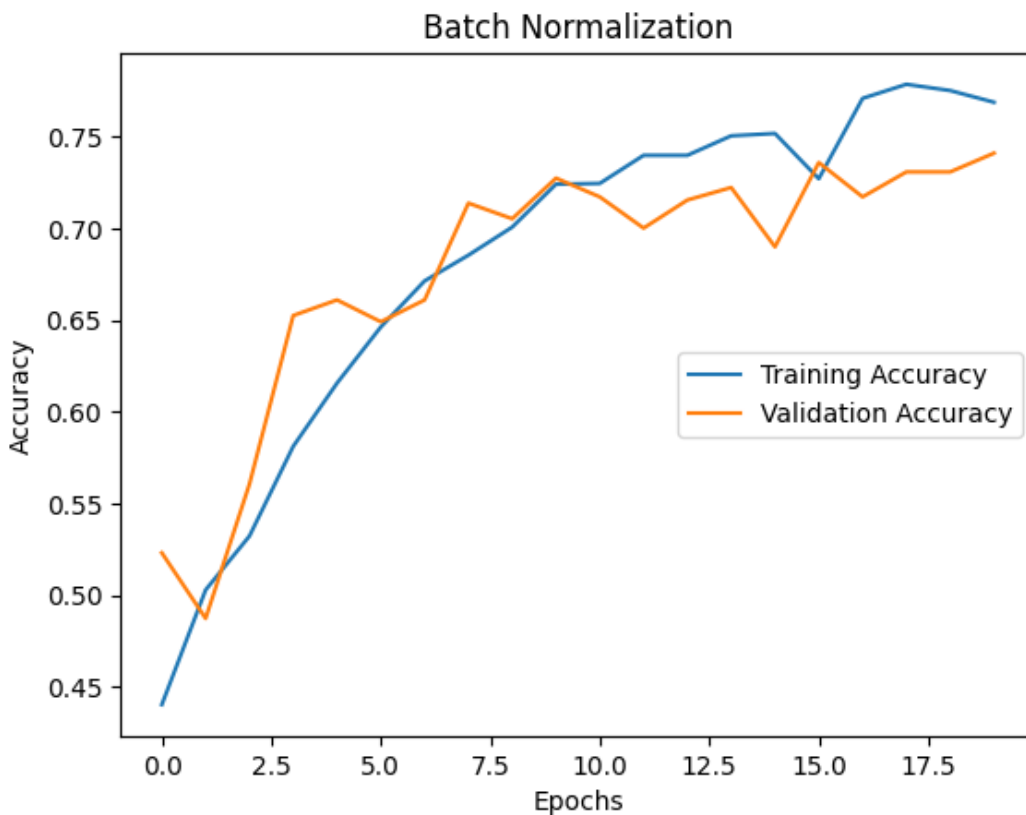
Epoch 20/20

74/74 ————— 42s 557ms/step - accuracy: 0.7837 - loss: 0.5893 - val_accu

y: 0.7411 - val_loss: 0.7955

In [150]:

```
# Creating dataframe
history_df = pd.DataFrame(history51.history)
# Plotting accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.title("Batch Normalization")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.savefig("Batch Normalization")
plt.show()
```



Stride of 2 and padding = same

In [121]:

```
model61 = Sequential()
# First Convolutional and pooling layer
model61.add(Conv2D(filters = 32, kernel_size = (3, 3), strides = (2, 2), padding = "same",
, input_shape = (128, 128, 3), activation = 'relu'))
model61.add(MaxPooling2D(pool_size = (2, 2), padding = "same"))
# Second Convolutional and pooling layer
model61.add(Conv2D(filters = 64, kernel_size = (3, 3), strides = (2, 2), padding = "same",
, activation = 'relu'))
model61.add(MaxPooling2D(pool_size = (2, 2), padding = "same"))
# Third Convolutional and pooling layer
model61.add(Conv2D(filters = 128, kernel_size = (3, 3), strides = (2, 2), padding = "same",
, activation = 'relu'))
model61.add(MaxPooling2D(pool_size = (2, 2), padding = "same"))
# Adding Flatten Layer
model61.add(Flatten())
# First dense layer with 256 neurons
```

```

model61.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model61.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model61.add(Dense(5, activation = 'softmax'))

#Compilation
model61.compile(optimizer="Adam", loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 32
history61 = model61.fit(train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                             batch_size = batch_size,
                                             subset = "validation"))

```

```

Epoch 1/20
74/74 ██████████ 16s 184ms/step - accuracy: 0.3127 - loss: 1.4969 - val_accu
racy: 0.5128 - val_loss: 1.1667
Epoch 2/20
74/74 ██████████ 14s 178ms/step - accuracy: 0.4879 - loss: 1.1823 - val_accu
racy: 0.5639 - val_loss: 1.1257
Epoch 3/20
74/74 ██████████ 16s 209ms/step - accuracy: 0.5465 - loss: 1.1242 - val_accu
racy: 0.5997 - val_loss: 0.9966
Epoch 4/20
74/74 ██████████ 14s 175ms/step - accuracy: 0.5759 - loss: 1.0151 - val_accu
racy: 0.6695 - val_loss: 0.8978
Epoch 5/20
74/74 ██████████ 14s 181ms/step - accuracy: 0.6183 - loss: 0.9389 - val_accu
racy: 0.6457 - val_loss: 0.9670
Epoch 6/20
74/74 ██████████ 14s 178ms/step - accuracy: 0.6387 - loss: 0.9056 - val_accu
racy: 0.6934 - val_loss: 0.8340
Epoch 7/20
74/74 ██████████ 14s 178ms/step - accuracy: 0.6622 - loss: 0.8423 - val_accu
racy: 0.6746 - val_loss: 0.8426
Epoch 8/20
74/74 ██████████ 15s 184ms/step - accuracy: 0.6917 - loss: 0.8164 - val_accu
racy: 0.7019 - val_loss: 0.8087
Epoch 9/20
74/74 ██████████ 17s 211ms/step - accuracy: 0.6939 - loss: 0.7599 - val_accu
racy: 0.6985 - val_loss: 0.7946
Epoch 10/20
74/74 ██████████ 15s 189ms/step - accuracy: 0.7043 - loss: 0.7614 - val_accu
racy: 0.7274 - val_loss: 0.7730
Epoch 11/20
74/74 ██████████ 14s 182ms/step - accuracy: 0.7170 - loss: 0.7439 - val_accu
racy: 0.7291 - val_loss: 0.7335
Epoch 12/20
74/74 ██████████ 16s 204ms/step - accuracy: 0.7183 - loss: 0.7151 - val_accu
racy: 0.6968 - val_loss: 0.8370
Epoch 13/20
74/74 ██████████ 17s 213ms/step - accuracy: 0.7289 - loss: 0.6887 - val_accu
racy: 0.7223 - val_loss: 0.7352
Epoch 14/20
74/74 ██████████ 15s 192ms/step - accuracy: 0.7443 - loss: 0.6619 - val_accu
racy: 0.7274 - val_loss: 0.7561
Epoch 15/20
74/74 ██████████ 15s 188ms/step - accuracy: 0.7398 - loss: 0.6631 - val_accu
racy: 0.7274 - val_loss: 0.7505
Epoch 16/20
74/74 ██████████ 16s 200ms/step - accuracy: 0.7487 - loss: 0.6497 - val_accu
racy: 0.7462 - val_loss: 0.7502
Epoch 17/20
74/74 ██████████ 15s 193ms/step - accuracy: 0.7636 - loss: 0.6273 - val_accu
racy: 0.7291 - val_loss: 0.7377
Epoch 18/20
74/74 ██████████ 14s 178ms/step - accuracy: 0.7726 - loss: 0.6094 - val_accu
racy: 0.7462 - val_loss: 0.7631

```


Epoch 19/20

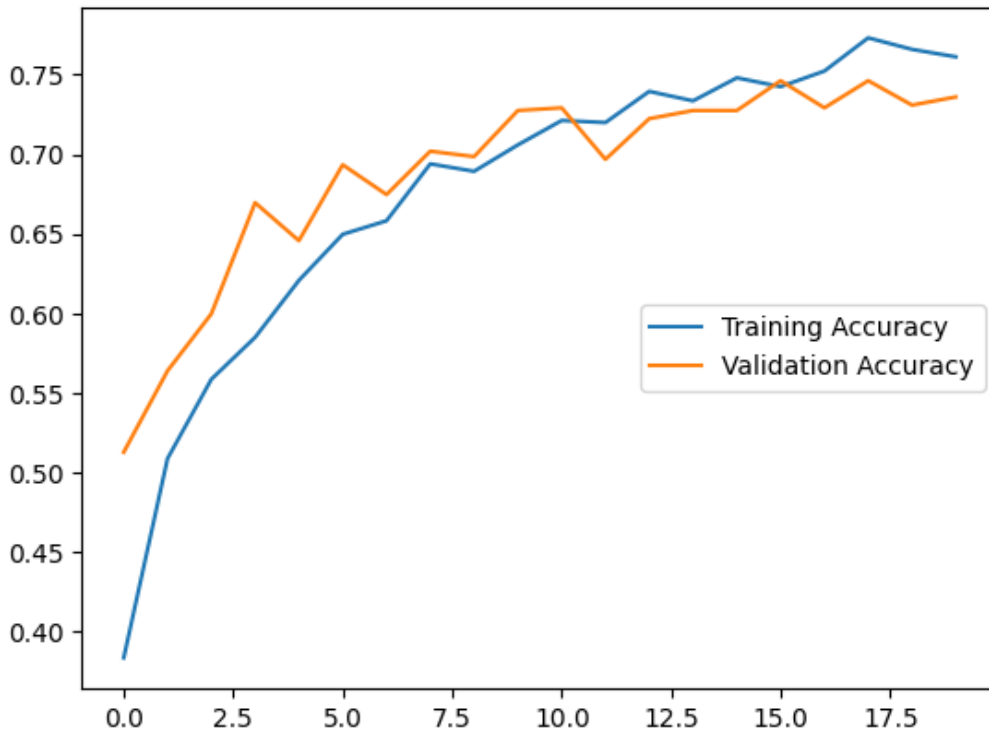
74/74 15s 184ms/step - accuracy: 0.7556 - loss: 0.6309 - val_accuracy: 0.7308 - val_loss: 0.7430

Epoch 20/20

74/74 14s 175ms/step - accuracy: 0.7613 - loss: 0.6115 - val_accuracy: 0.7359 - val_loss: 0.7406

In [151]:

```
# Creating dataframe
history_df = pd.DataFrame(history61.history)
# Plotting accuracy
plt.plot(history_df["accuracy"], label = "Training Accuracy")
plt.plot(history_df["val_accuracy"], label = "Validation Accuracy")
plt.legend(loc=7)
plt.savefig("Kernel stride padding batch normalization")
plt.show()
```



Comparison of validation accuracy for different units in FC

In [127]:

```
neurons = [64, 128, 256, 512]
validation_accuracy = [0.7274, 0.7462, 0.7888, 0.7836]

# Plotting
plt.figure(figsize=(10, 6))
bars = plt.bar(range(len(neurons)), validation_accuracy, color='skyblue', width=0.4, edge
color = "black")

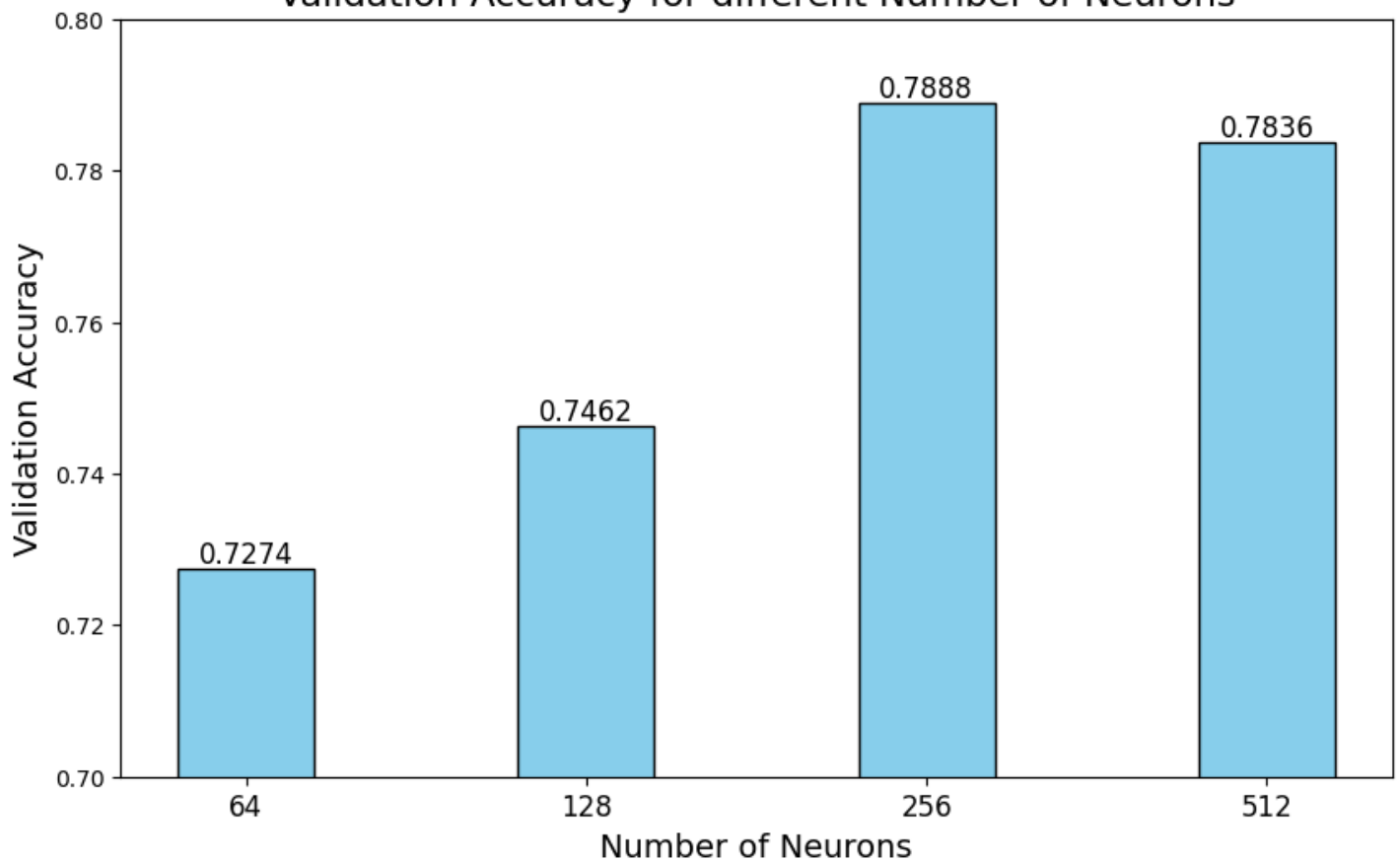
for i, bar in enumerate(bars):
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.4f}', ha='center', va='bo
ttom', fontsize=12)

plt.xticks(range(len(neurons)), neurons, fontsize=12)

plt.title('Validation Accuracy for different Number of Neurons', fontsize=16)
plt.xlabel('Number of Neurons', fontsize=14)
plt.ylabel('Validation Accuracy', fontsize=14)
plt.ylim(0.7, 0.8)
plt.savefig("Different neurons")
plt.show()
```

Validation Accuracy for different Number of Neurons

Validation Accuracy for different Number of Neurons

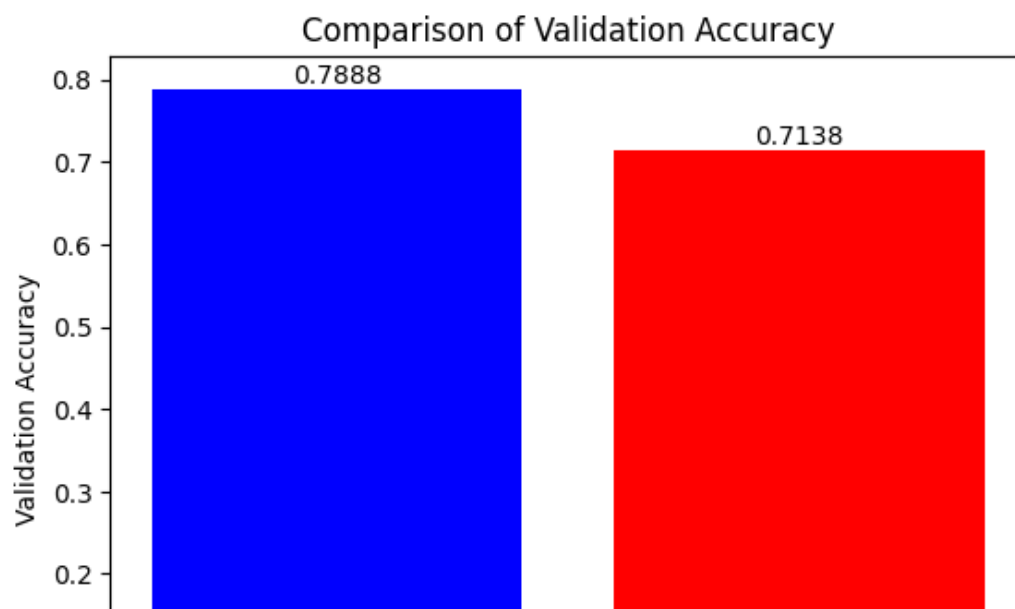


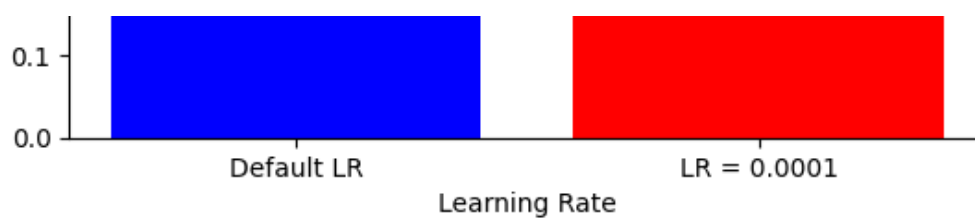
Comparison Of validation accuracy of default learning rate & 0.0001 learning rate

In [116]:

```
learning_rates = ['Default LR', 'LR = 0.0001']
validation_accuracies = [0.7888, 0.7138]

# Plotting
bars = plt.bar(learning_rates, validation_accuracies, color=['blue', 'red'])
plt.title('Comparison of Validation Accuracy')
plt.xlabel('Learning Rate')
plt.ylabel('Validation Accuracy')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.4f}', ha='center', va='bottom')
plt.savefig("Comparison of learning rates")
plt.show()
```





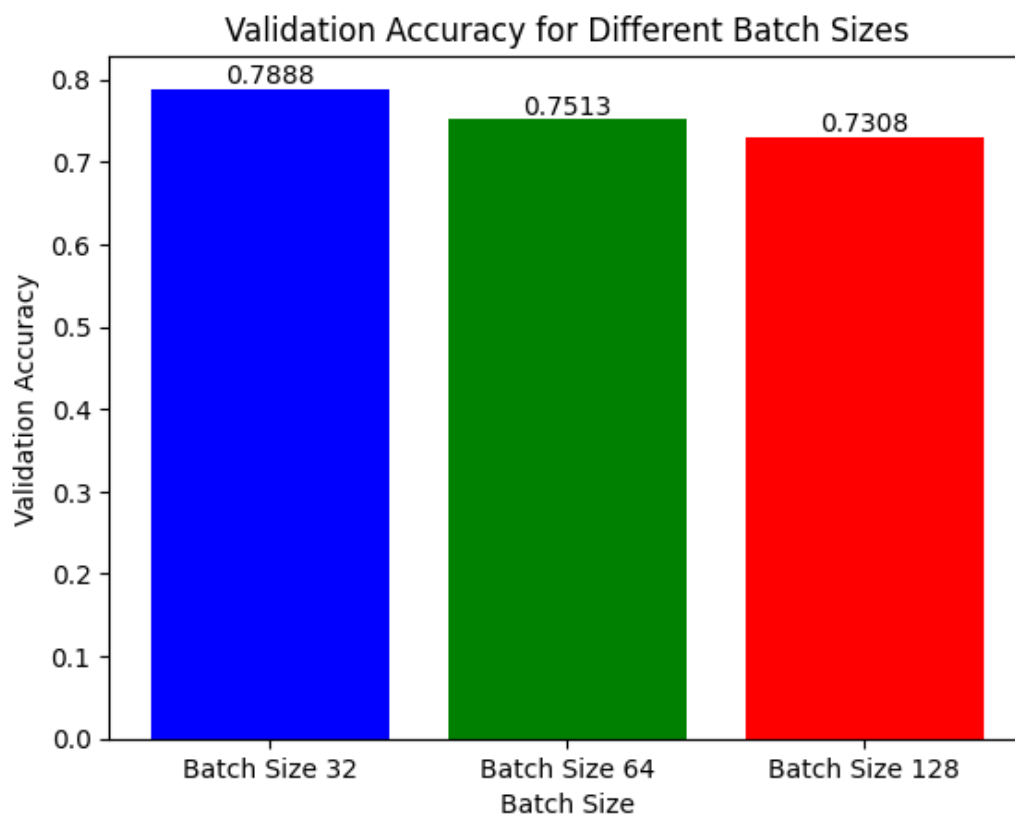
Comparison between different batch size

In [118]:

```
batch_sizes = ['Batch Size 32', 'Batch Size 64', 'Batch Size 128']
validation_accuracies = [0.7888, 0.7513, 0.7308]

# Plotting
bars = plt.bar(batch_sizes, validation_accuracies, color=['blue', 'green', 'red'])
plt.title('Validation Accuracy for Different Batch Sizes')
plt.xlabel('Batch Size')
plt.ylabel('Validation Accuracy')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.4f}', ha='center', va='bottom')

plt.savefig("Comparison between batch size")
plt.show()
```



Comparison of different Kernel Size

In [124]:

```
kernel_sizes = ['Kernel 3x3', 'Kernel 5x5']
validation_accuracies = [0.7888, 0.7376]

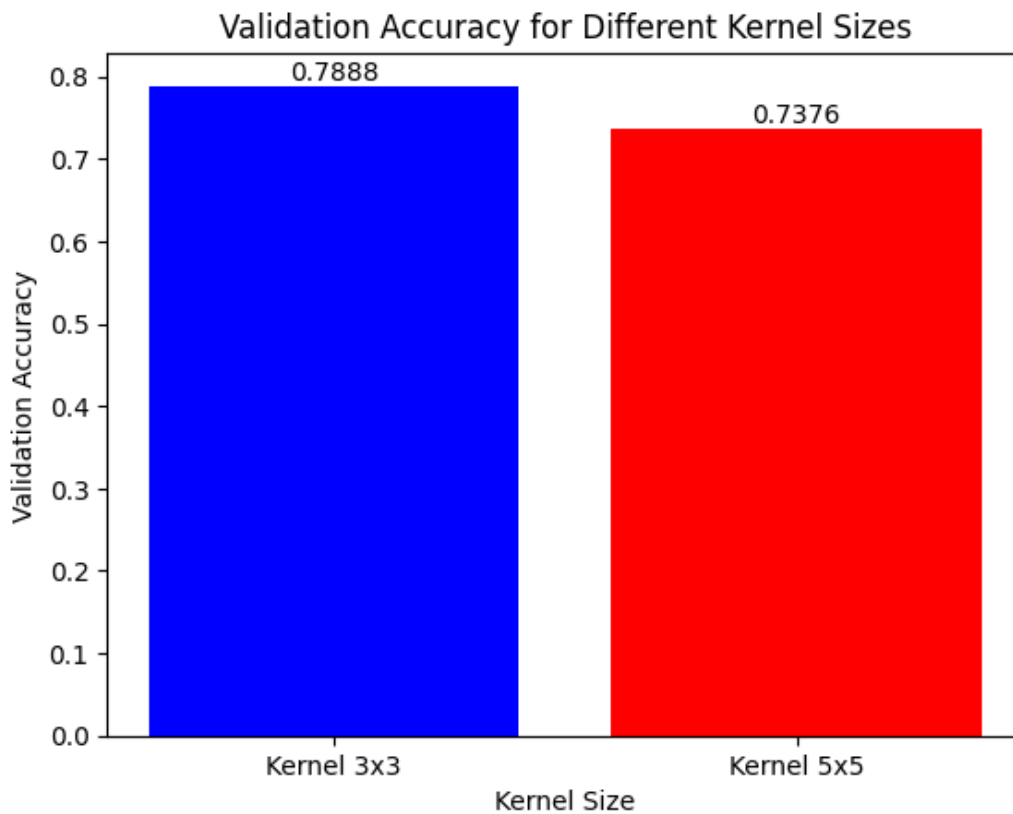
# Plotting
bars = plt.bar(kernel_sizes, validation_accuracies, color=['blue', 'red'])
plt.title('Validation Accuracy for Different Kernel Sizes')
plt.xlabel('Kernel Size')
plt.ylabel('Validation Accuracy')
```

```

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.4f}', ha='center', va='bottom')

plt.savefig("Comparison of different Kernel Size")
plt.show()

```



Comparison of Batch Normalization

In [126]:

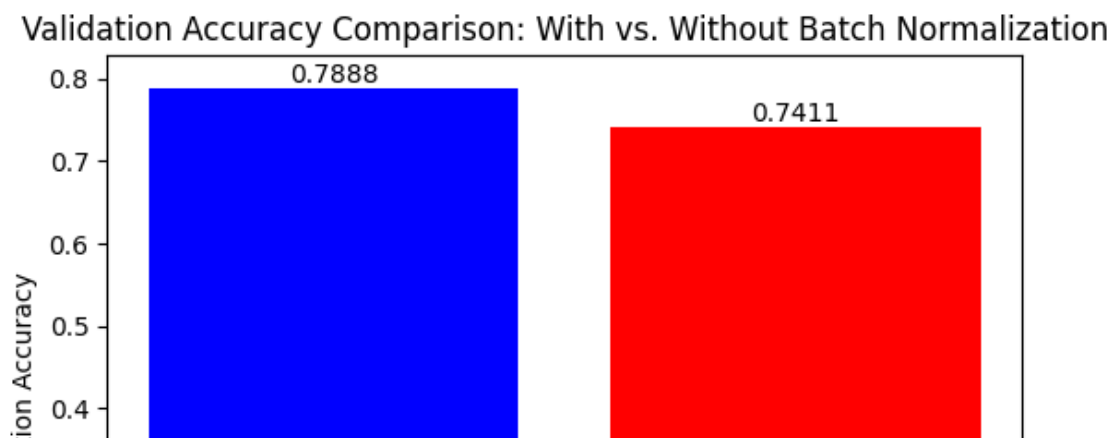
```

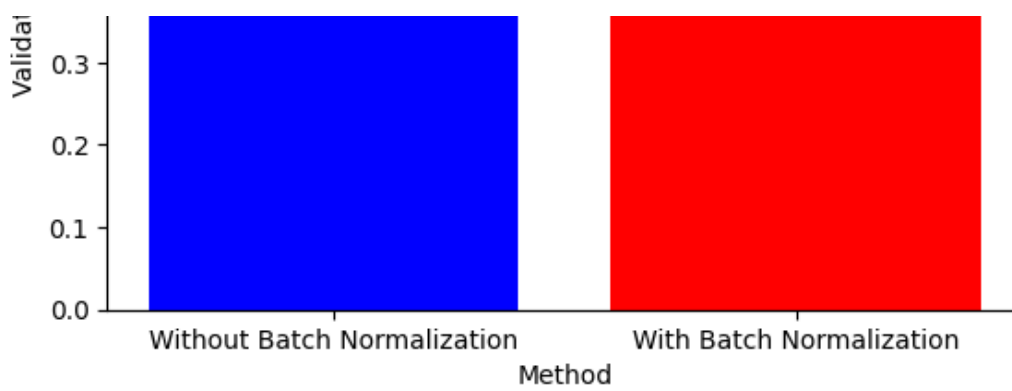
methods = ['Without Batch Normalization', 'With Batch Normalization']
validation_accuracies = [0.7888, 0.7411]

# Plotting
bars = plt.bar(methods, validation_accuracies, color=['blue', 'red'])
plt.title('Validation Accuracy Comparison: With vs. Without Batch Normalization')
plt.xlabel('Method')
plt.ylabel('Validation Accuracy')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.4f}', ha='center', va='bottom')

plt.savefig("Batch Normalization")
plt.show()

```





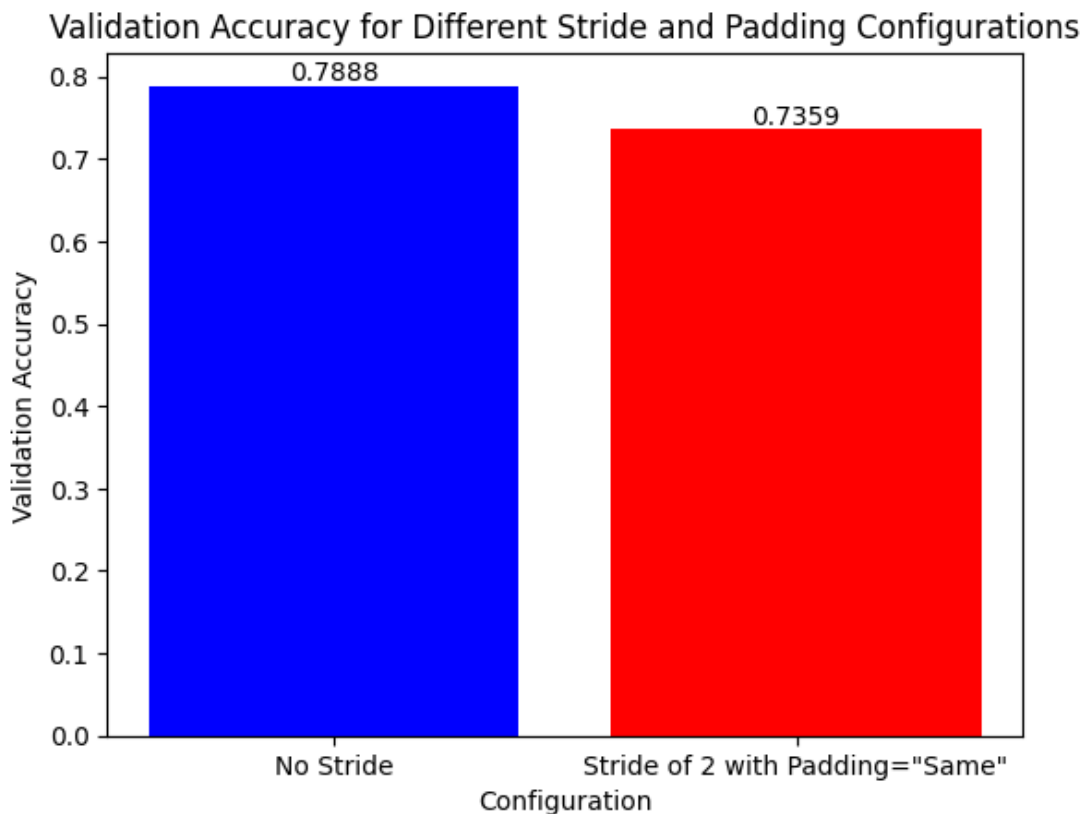
Effect of stride and padding

In [129]:

```
configurations = ['No Stride', 'Stride of 2 with Padding="Same"']
validation_accuracies = [0.7888, 0.7359]

# Plotting
bars = plt.bar(configurations, validation_accuracies, color=['blue', 'red'])
plt.title('Validation Accuracy for Different Stride and Padding Configurations')
plt.xlabel('Configuration')
plt.ylabel('Validation Accuracy')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.4f}', ha='center', va='bottom')

plt.savefig("Stride & Padding")
plt.show()
```



In []: