Name: Ariyan Hossain
ID: 20101099
Sec: 01
Course: CSE425

Assignment 2: Exploring Traditional CNNs with Different Activation Functions and Convolutional Layers

1. Dataset Selection:

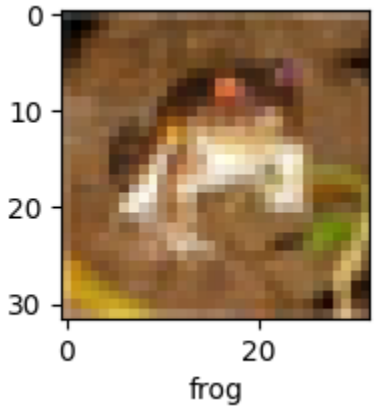
The CIFAR-10 dataset (Canadian Institute for Advanced Research, 10 classes) is used which is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The images are labeled with one of 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck). There are 6000 images per class with 5000 training and 1000 testing images per class.

The dataset is loaded from Keras.

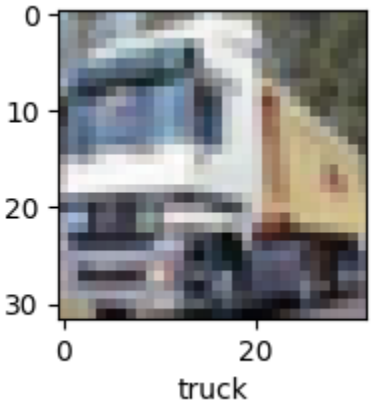
```
from tensorflow.keras import datasets
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
```

X_train Shape: (50000, 32, 32, 3)
X_test Shape: (10000, 32, 32, 3)
y_train Shape: (50000, 1)
y_test Shape: (10000, 1)

X_train[0]:



X_train[1]:



For normalizing the images to a number from 0 to 1, we divided it by 255 as images have 3 channels (R,G,B) and each value in the channel can range from 0 to 255.

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

2. Implementing Traditional CNNs:

For implementing a CNN model, two bundles of convolution layer and max pooling layer are used. First, a convolution layer with 32 filters, 3x3 kernel size and stride 1 is used and then it is passed to a max pooling of 2x2 layer which is the most common. It is then passed to another similar convolution layer and max pooling layer. Finally, the outputs are flattened and passed to a fully connected network having 64 neurons in the first layer and 10 neurons as the last output layer since there are 10 classes. For the output layer, “softmax” activation is used to convert the outputs into probabilities for classifying the images into the 10 classes. For the other layers, “relu” activation is used.

```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu',
    input_shape=(32, 32, 3)),
   layers.MaxPooling2D((2, 2)),

   layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
   layers.MaxPooling2D((2, 2)),

   layers.Flatten(),
   layers.Dense(64, activation='relu'),
   layers.Dense(10, activation='softmax')
])
```

For training, “adam” optimizer is used which is the most commonly used gradient descent-based optimization algorithm. For calculating loss, “sparse_categorical_crossentropy” is used which is generally used for classification tasks where the target variable is categorical. In this task, for the 10 different classes, each class is represented by a number from 0 to 9. The categorical cross-entropy loss is a measure of the difference between the predicted probabilities of the classes and the true class labels. The epochs for training the model is set as 10.

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

cnn.fit(X_train, y_train, epochs=10)
```

3. Activation Functions:

For activation functions, 3 different activation functions are used in training the model and compared. “relu”, “tanh”, “sigmoid” are used in this task.

ReLU (Rectified Linear Activation): Outputs the input if it's positive, otherwise outputs zero.
Tanh (Hyperbolic Tangent Activation): Scales input to range between -1 and 1, centered at 0.
Sigmoid Activation: Squeezes input to range between 0 and 1, often used for probabilities.

```
activation_functions = ['relu', 'tanh', 'sigmoid']
for activation in activation_functions:
        cnn_model = models.Sequential([
            layers.Conv2D(filters=num_filters, kernel_size=kernel_size, strides=stride,
            activation=activation, input_shape=(32, 32, 3)),
            layers.MaxPooling2D((2, 2)),
            layers.Conv2D(filters=num_filters, kernel_size=kernel_size, strides=stride,
            activation=activation),
            layers.MaxPooling2D((2, 2)),
            layers.Flatten(),
            layers.Dense(64, activation=activation),
            layers.Dense(10, activation='softmax')
        ])
```

4. Convolutional Layer Configurations:

Different convolutional layer configurations with varying the number of filters, kernel sizes, and strides are used while keeping the dataset and hyperparameters fixed to have a complete comparison. Number of filters of 32 and 64 are used, kernel sizes of 3x3 and 5x5 are used and strides of 1 and 2 are used.

```
combinations = [
    ("relu", 32, (3, 3), (1, 1)),
    ("relu", 64, (3, 3), (1, 1)),
    ("relu", 32, (5, 5), (1, 1)),
    ("relu", 32, (3, 3), (2, 2)),

    ("tanh", 32, (3, 3), (1, 1)),
    ("tanh", 64, (3, 3), (1, 1)),
    ("tanh", 32, (5, 5), (1, 1)),
    ("tanh", 32, (3, 3), (2, 2)),
```

```
    ("sigmoid", 32, (3, 3), (1, 1)),
    ("sigmoid", 64, (3, 3), (1, 1)),
    ("sigmoid", 32, (5, 5), (1, 1)),
    ("sigmoid", 32, (3, 3), (2, 2))
]


# Loop over combinations
for activation, num_filters, kernel_size, stride in combinations:
    # Create the CNN model
    cnn_model = models.Sequential([
        layers.Conv2D(filters=num_filters, kernel_size=kernel_size, strides=stride,
        activation=activation, input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(filters=num_filters, kernel_size=kernel_size, strides=stride,
       activation=activation),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(64, activation=activation),
        layers.Dense(10, activation='softmax')
    ])

    # Compile the model
    cnn_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Train the model
    cnn_model.fit(X_train, y_train, epochs=10, verbose=0)
```

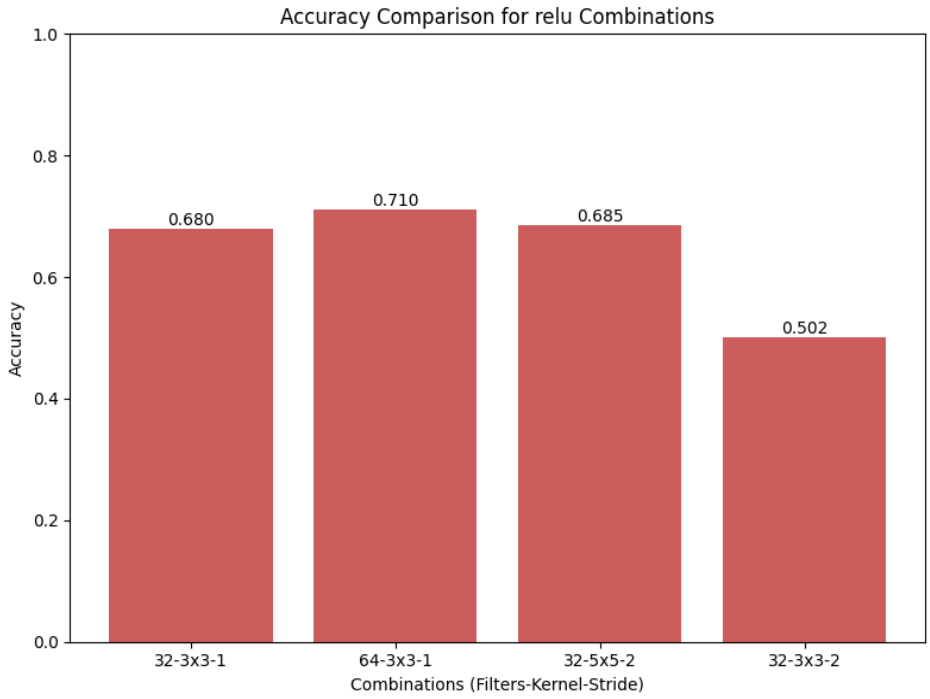
To evaluate on test data:

```
loss, accuracy = cnn_model.evaluate(X_test, y_test)
```

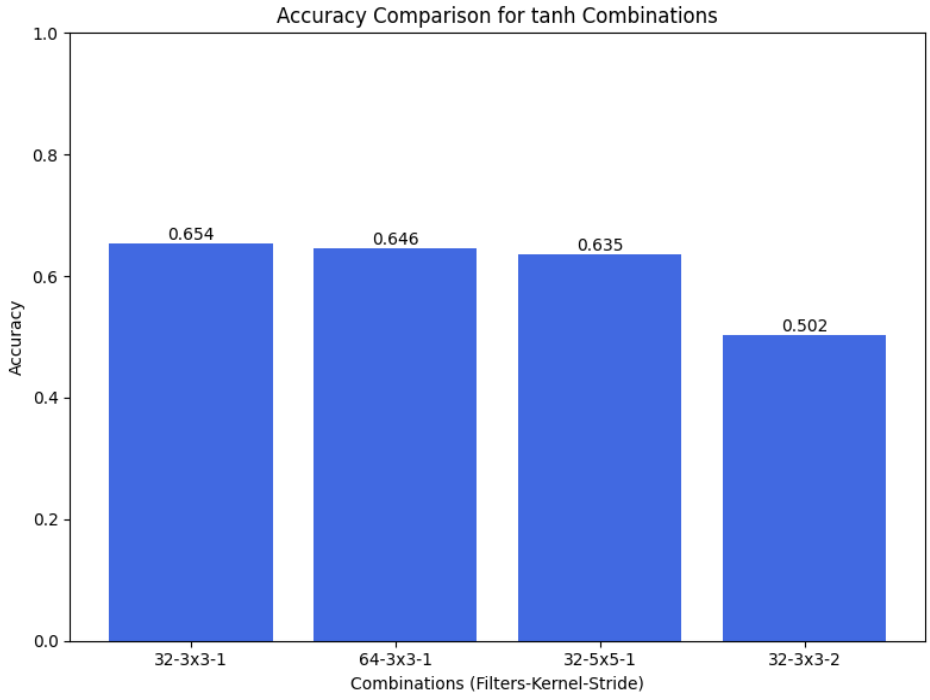
5. Performance Evaluation:

activation	filters	kernel	stride	accuracy	macro avg precision	macro avg recall	macro avg f1-score
relu	32	(3, 3)	(1, 1)	0.680	0.682	0.680	0.675
relu	64	(3, 3)	(1, 1)	0.710	0.709	0.710	0.709
relu	32	(5, 5)	(1, 1)	0.685	0.689	0.685	0.683
relu	32	(3, 3)	(2, 2)	0.502	0.513	0.502	0.504
tanh	32	(3, 3)	(1, 1)	0.654	0.654	0.654	0.652
tanh	64	(3, 3)	(1, 1)	0.646	0.645	0.646	0.642
tanh	32	(5, 5)	(1, 1)	0.635	0.635	0.635	0.632
tanh	32	(3, 3)	(2, 2)	0.502	0.510	0.502	0.501
sigmoid	32	(3, 3)	(1, 1)	0.580	0.58	0.580	0.575
sigmoid	64	(3, 3)	(1, 1)	0.601	0.601	0.601	0.597
sigmoid	32	(5, 5)	(1, 1)	0.586	0.597	0.586	0.585
sigmoid	32	(3, 3)	(2, 2)	0.382	0.395	0.382	0.374

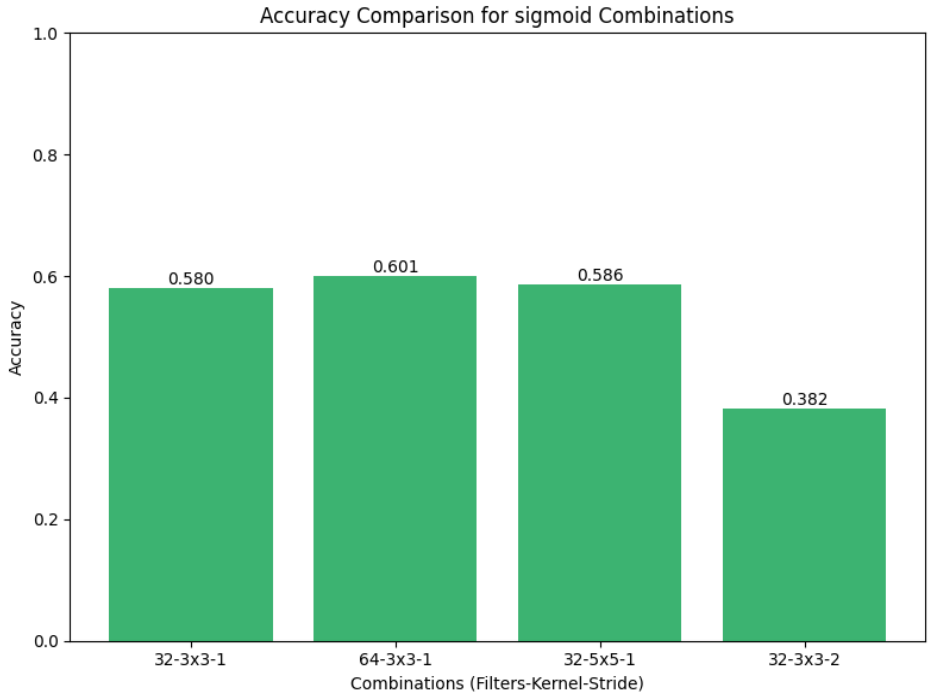
Different combinations of filters-kernel-stride used to compare using relu activation:



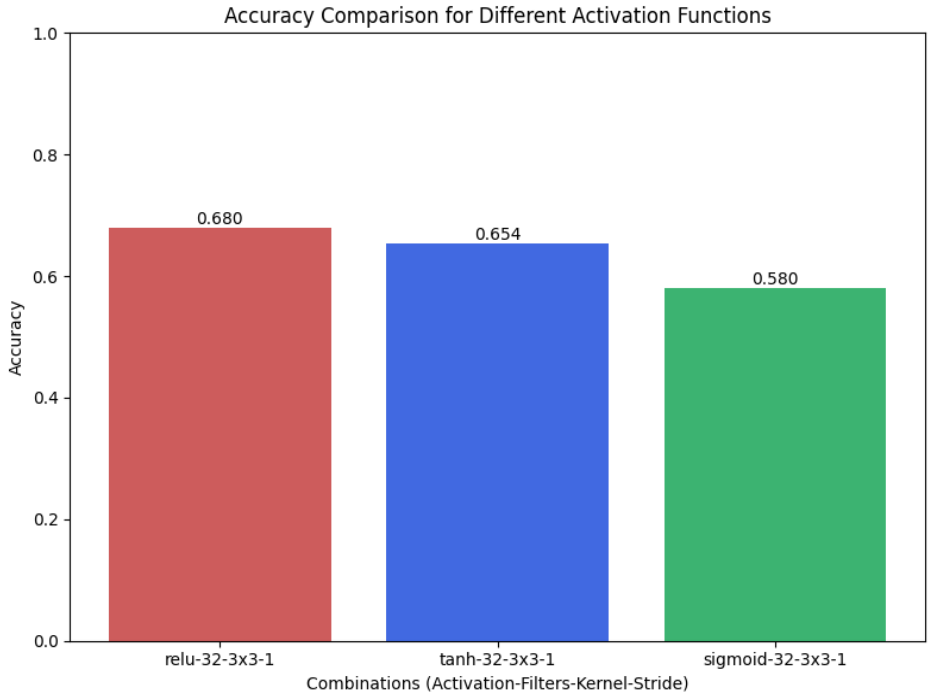
Different combinations of filters-kernel-stride used to compare using tanh activation:



Different combinations of filters-kernel-stride used to compare using sigmoid activation:



Using a fixed combination of filters-kernel-stride, different activation functions are compared:



6. Discussion:

Activation Functions:

ReLU:

Strengths: The simplicity of thresholding allows for rapid computing; the vanishing gradient problem is overcome; and the method is effective in most situations.
Weaknesses: Can suffer from the "dying ReLU" problem where neurons can become inactive during training.

Tanh:

Strengths: Convergence may be aided by the fact that the outputs vary from -1 to 1, with zero as the center, and the dying ReLU problem is avoided.
Weaknesses: Not necessarily as computationally efficient as ReLU, and still vulnerable to vanishing gradients.

Sigmoid:

Strengths: The outputs are condensed between 0 and 1, making them well-suited for binary classification and probability-like outcomes.
Weaknesses: Prone to vanishing gradient, outputs not centered at 0 (can lead to slower convergence), not commonly used in hidden layers due to gradient issues.

Convolutional Layer Setup:

Number of Filters:

Strengths: Increasing filters can capture more complex features, leading to better representation.
Weaknesses: Using an excessive number of filters might increase computation time and, if not regularized, can lead to overfitting.

Kernel Size:

Strengths: More detailed patterns are captured by smaller kernels whereas bigger patterns are captured by smaller kernels.
Weaknesses: Small kernels may exclude crucial features whereas large kernels increase processing and parameters.

Stride:

Strengths: Larger strides result in a reduction in computation and spatial dimensions, which can aid in downsampling.
Weaknesses: Information loss from large strides is possible, especially in the early layers..

It can be observed from the graphs that for a fixed activation function

- As number of filters increases, accuracies increases (0.680 to 0.710 for relu)
- As kernel size increases, accuracies slightly increases (0.680 to 0.685 for relu)
- As stride increases, accuracies decreases (0.680 to 0.502 for relu)

It can also be observed that for a fixed combination of filters-kernel-stride

- ReLU gives the most accuracy (0.680 for filters=32, kernel=3x3, stride=1)
- Tanh gives a lower accuracy (0.654 for filters=32, kernel=3x3, stride=1)
- Sigmoid gives the lowest accuracy (0.580 for filters=32, kernel=3x3, stride=1)

7. Conclusion and Future Work:

The effects of several activation functions, filter sizes, kernel sizes, and strides on the model's performance in this CNN training experiment were investigated. On a particular dataset, the models were trained and assessed a number of CNN architectures, assessing their accuracy, macro-averaged precision, recall, and F1-score. It was learned through this trial how different settings affect the network's capacity to recognise and generalize data patterns.

Primary Findings:

The convergence and effectiveness of the CNN are greatly impacted by activation functions. Tanh and sigmoid both showed efficacy in some cases, while ReLU also displayed competitive performance with quick convergence. The capacity of the model to capture complex characteristics depends on the number of filters. The network's representation capability can be increased by adding filters, but there is a trade-off with increased computational complexity. The receptive field of the network is impacted by kernel size. While smaller kernels concentrate on finer details, larger kernels are better at capturing larger patterns. For best performance, a balance must be established between these. The downsampling rate and information preservation are influenced by stride. To avoid losing significant characteristics, stride values must be carefully chosen.

Future Work:

Hyperparameter Tuning: Additional fine-tuning of hyperparameters, such as learning rate, regularization, and batch size, can produce better outcomes.

Data Augmentation: Use data augmentation techniques to inflate the dataset's size artificially, which might improve generalization and resilience.

Transfer Learning: Investigate transfer learning by modifying pre-trained models, such those from ImageNet, for a particular dataset.

Regularization: To avoid overfitting and improve model generalization, try out various regularization strategies like batch normalization or dropout.

Architecture Variations: To capture even more complicated patterns, experimenting with more sophisticated structures like residual networks (ResNets) or inception networks.

We can improve CNN's performance, comprehend its behavior on various datasets, and contribute to the continued improvement of convolutional neural networks in the area of deep learning by tackling these future topics.

References:

Sharma, P. (2022). Basic Introduction to Convolutional Neural Network in Deep Learning. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/

Image classification using CNN: Introduction and tutorial. (2023, May 22). Datagen. https://datagen.tech/guides/image-classification/image-classification-using-cnn/#