Submitted By: Mrinal Bhan *DSAI 211020428*

1) Load CIFAR-100 dataset and Apply Convolutional Neural Network (CNN) to train and compile the model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import math
import datetime
import platform


```

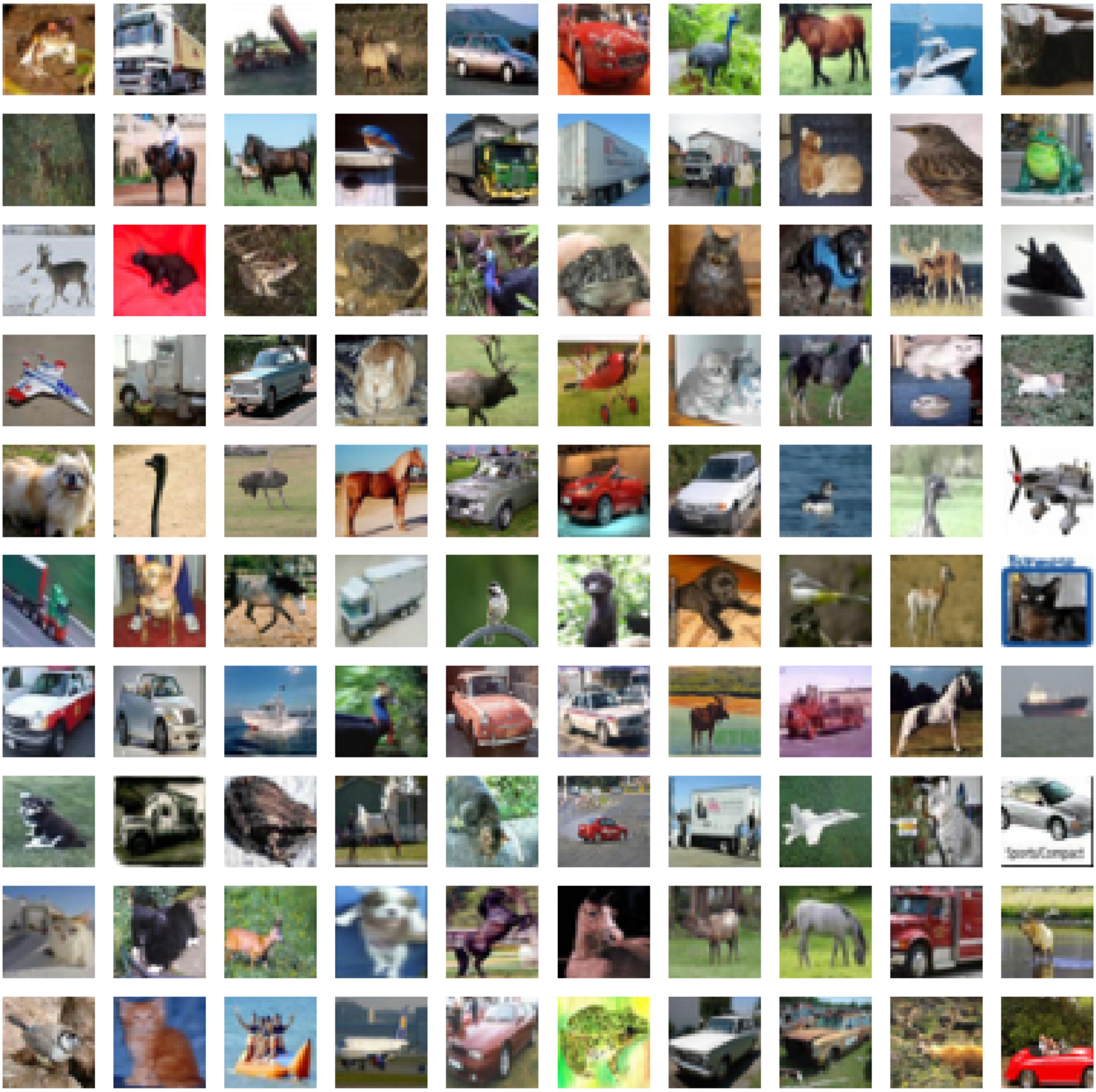
```
tf.keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
((array([[[[ 59,  62,  63],
           [ 43,  46,  45],
           [ 50,  48,  43],
           ...,
           [158, 132, 108],
           [152, 125, 102],
           [148, 124, 103]],

          [[ 16,  20,  20],
           [  0,   0,   0],
           [ 18,   8,   0],
           ...,
           [123,  88,  55],
           [119,  83,  50],
           [122,  87,  57]],

          [[ 25,  24,  21],
           [ 16,   7,   0],
           [ 49,  27,   8],
           ...,
           [118,  84,  50],
           [120,  84,  50],
           [109,  73,  42]],

          ...,

          [[208, 170,  96],
           [201, 153,  34],
           [198, 161,  26],
           ...,
           [160, 133,  70],
           [ 56,  31,   7],
           [ 53,  34,  20]],

          [[180, 139,  96],
           [173, 123,  42],
           [186, 144,  30],
           ...,
           [184, 148,  94],
           [ 97,  62,  34],
           [ 83,  53,  34]],

          [[177, 144, 116],
           [168, 129,  94],
           [179, 142,  87],
           ...,
           [216, 184, 140],
           [151, 118,  84],
           [123,  92,  72]]],


         [[[154, 177, 187],
           [126, 137, 136],
           [105, 104,  95],
           ...,
           [ 91,  95,  71],
```

```
(train_images, train_labels), (test_images, test_labels) = keras.datasets.cifar10.load_data()
```

```
plt.figure(figsize = (16,16))
for i in range(100):
  plt.subplot(10,10,1+i)
```

```
plt.axis('off')
plt.imshow(train_images[i], cmap = 'gray')
```



```
# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.activations import relu, sigmoid, tanh
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import datasets, layers, models
```

```
# Define the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.MaxPooling2D((2, 2)))

# Add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

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

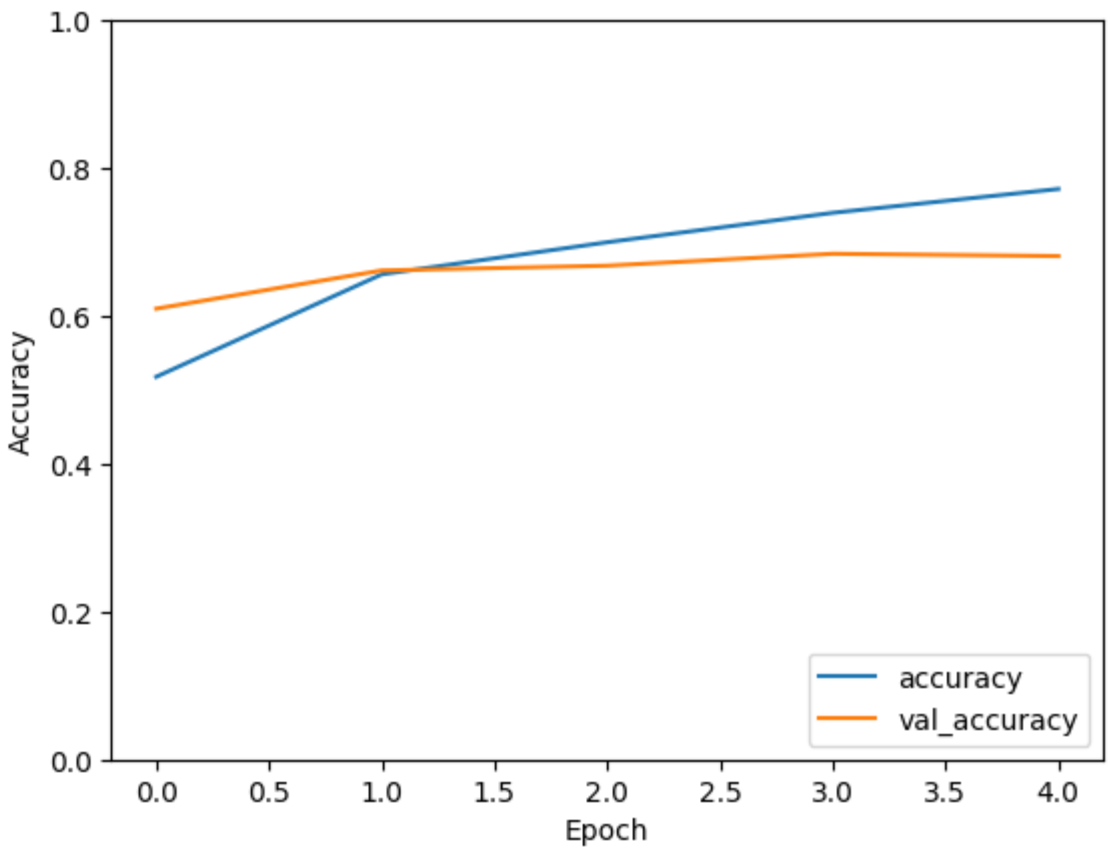
```
# Train the model
history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/5
1563/1563 [==============================] - 19s 5ms/step - loss: 1.3589 - accuracy: 0.5176 - val_loss: 1.1224 - val_accuracy: 0.6098
Epoch 2/5
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9895 - accuracy: 0.6563 - val_loss: 0.9849 - val_accuracy: 0.6615
Epoch 3/5
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8593 - accuracy: 0.6997 - val_loss: 0.9683 - val_accuracy: 0.6680
Epoch 4/5
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7516 - accuracy: 0.7395 - val_loss: 0.9347 - val_accuracy: 0.6841
Epoch 5/5
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6571 - accuracy: 0.7717 - val_loss: 0.9518 - val_accuracy: 0.6810
```

```
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")
```

```
313/313 - 1s - loss: 0.9518 - accuracy: 0.6810 - 702ms/epoch - 2ms/step
Test accuracy: 0.6809999942779541
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



After simple CNN, we got 76% training accuracy and 69% testing accuracy

2) Apply different number of convolutional layer + kernel size + channel number to progress good accuracy.

Adding 3 Conv Layer

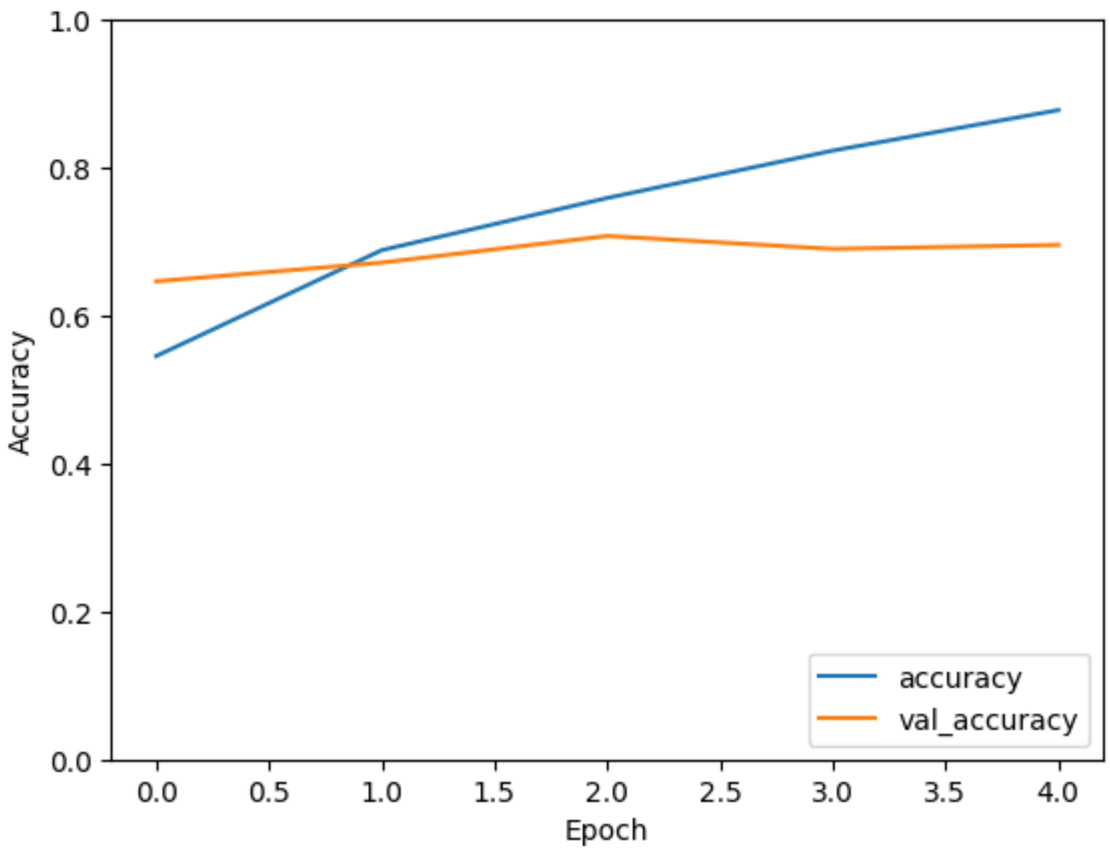
```
def cnn_build(kernel_size, channel):
  # Define the CNN model
  model = models.Sequential()

  # Convolutional layers with varying parameters
  model.add(layers.Conv2D(channel, (kernel_size, kernel_size), activation='relu', input_shape=(32, 32, 3)))
  model.add(layers.Conv2D(channel*2, (kernel_size, kernel_size), activation='relu'))
  model.add(layers.Conv2D(channel*4, (kernel_size, kernel_size), activation='relu'))
```

```
model.add(layers.MaxPooling2D((2, 2)))

# Flatten and add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))

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

# Train the model
history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
print("-----------------------------------")
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")
print("-----------------------------------")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
print()
print("-----------------------------------")
print()
```

Using 3 kernel size and 32 channels

```
1 cnn_build(3,32)
```

```
Epoch 1/5
1563/1563 [==============================] - 14s 8ms/step - loss: 1.2813 - accuracy: 0.5453 - val_loss: 1.0134 - val_accuracy: 0.6464
Epoch 2/5
1563/1563 [==============================] - 12s 8ms/step - loss: 0.8897 - accuracy: 0.6885 - val_loss: 0.9409 - val_accuracy: 0.6717
Epoch 3/5
1563/1563 [==============================] - 12s 8ms/step - loss: 0.6864 - accuracy: 0.7590 - val_loss: 0.8686 - val_accuracy: 0.7077
Epoch 4/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.5080 - accuracy: 0.8231 - val_loss: 0.9841 - val_accuracy: 0.6902
Epoch 5/5
1563/1563 [==============================] - 12s 8ms/step - loss: 0.3532 - accuracy: 0.8779 - val_loss: 1.0767 - val_accuracy: 0.6956
-----------------------------------
313/313 - 1s - loss: 1.0767 - accuracy: 0.6956 - 982ms/epoch - 3ms/step
Test accuracy: 0.6955999732017517
-----------------------------------
```



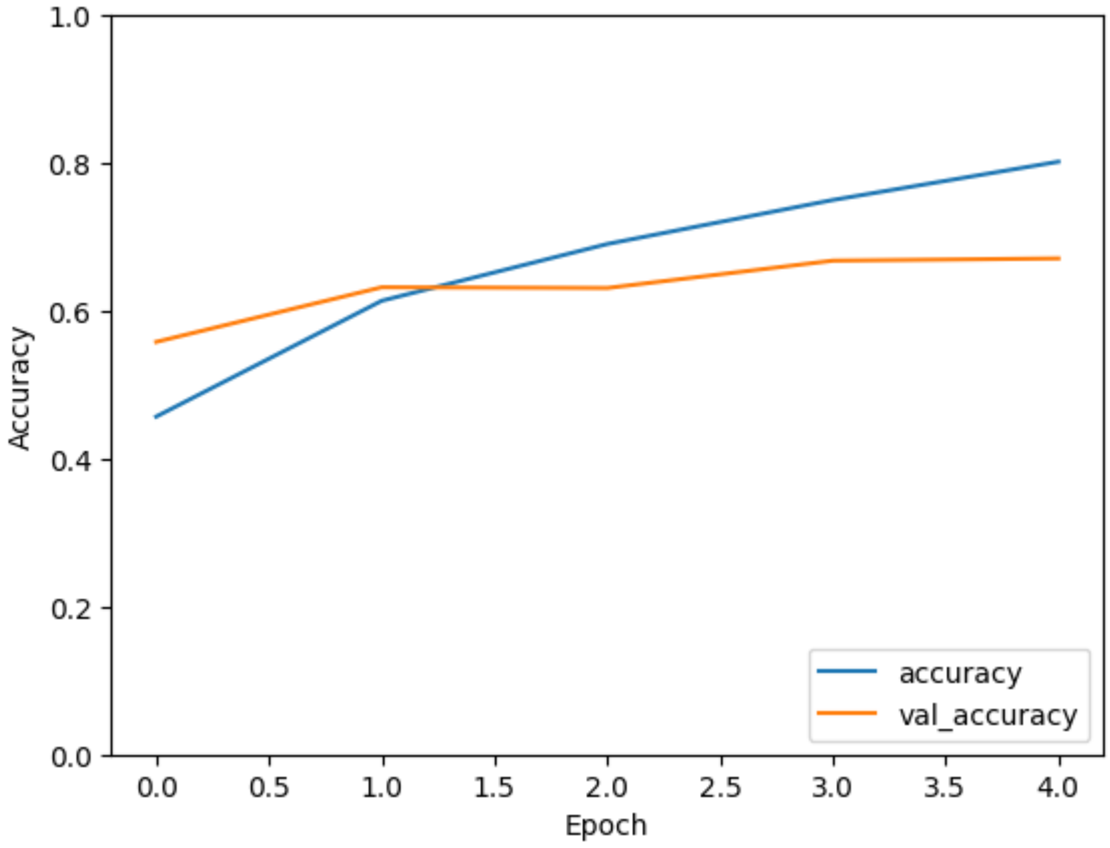

```
-----------------------------------
```

Using 5 kernel size and 48 channels

```
1 cnn_build(5,48)
```

```
Epoch 1/5
1563/1563 [==============================] - 18s 10ms/step - loss: 1.4937 - accuracy: 0.4570 - val_loss: 1.2547 - val_accuracy: 0.5581
```

```
Epoch 2/5
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0963 - accuracy: 0.6138 - val_loss: 1.0597 - val_accuracy: 0.6320
Epoch 3/5
1563/1563 [==============================] - 16s 10ms/step - loss: 0.8837 - accuracy: 0.6905 - val_loss: 1.0614 - val_accuracy: 0.6310
Epoch 4/5
1563/1563 [==============================] - 15s 10ms/step - loss: 0.7137 - accuracy: 0.7501 - val_loss: 1.0310 - val_accuracy: 0.6680
Epoch 5/5
1563/1563 [==============================] - 15s 10ms/step - loss: 0.5639 - accuracy: 0.8019 - val_loss: 1.0581 - val_accuracy: 0.6710
-----------------------------------
313/313 - 1s - loss: 1.0581 - accuracy: 0.6710 - 1s/epoch - 3ms/step
Test accuracy: 0.6710000038146973
-----------------------------------
```

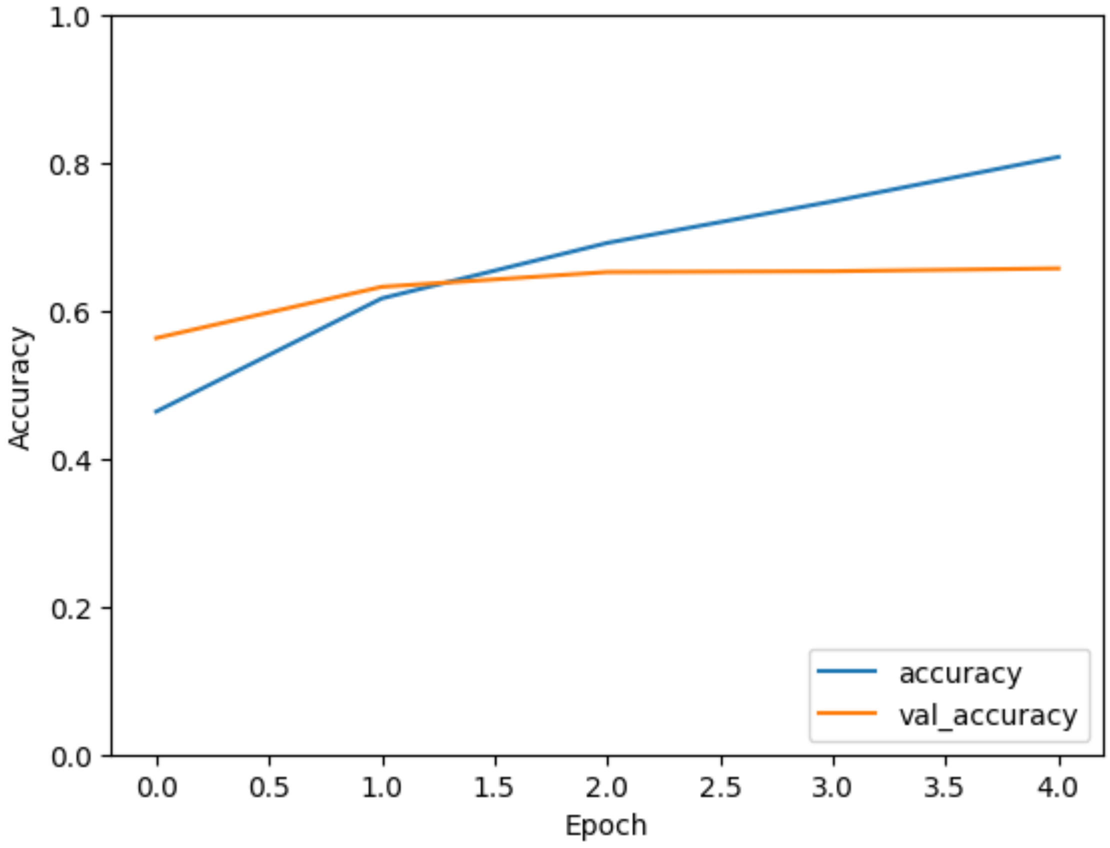



```
-----------------------------------
```

Using 5 kernel size and 32 channels

```
1 cnn_build(5,32)
```

```
Epoch 1/5
1563/1563 [==============================] - 15s 8ms/step - loss: 1.4813 - accuracy: 0.4638 - val_loss: 1.2233 - val_accuracy: 0.5633
Epoch 2/5
1563/1563 [==============================] - 11s 7ms/step - loss: 1.0850 - accuracy: 0.6170 - val_loss: 1.0533 - val_accuracy: 0.6326
Epoch 3/5
1563/1563 [==============================] - 11s 7ms/step - loss: 0.8785 - accuracy: 0.6920 - val_loss: 1.0097 - val_accuracy: 0.6526
Epoch 4/5
1563/1563 [==============================] - 11s 7ms/step - loss: 0.7134 - accuracy: 0.7483 - val_loss: 1.0447 - val_accuracy: 0.6537
Epoch 5/5
1563/1563 [==============================] - 11s 7ms/step - loss: 0.5507 - accuracy: 0.8083 - val_loss: 1.1146 - val_accuracy: 0.6575
-----------------------------------
313/313 - 1s - loss: 1.1146 - accuracy: 0.6575 - 1s/epoch - 4ms/step
Test accuracy: 0.6575000286102295
-----------------------------------
```

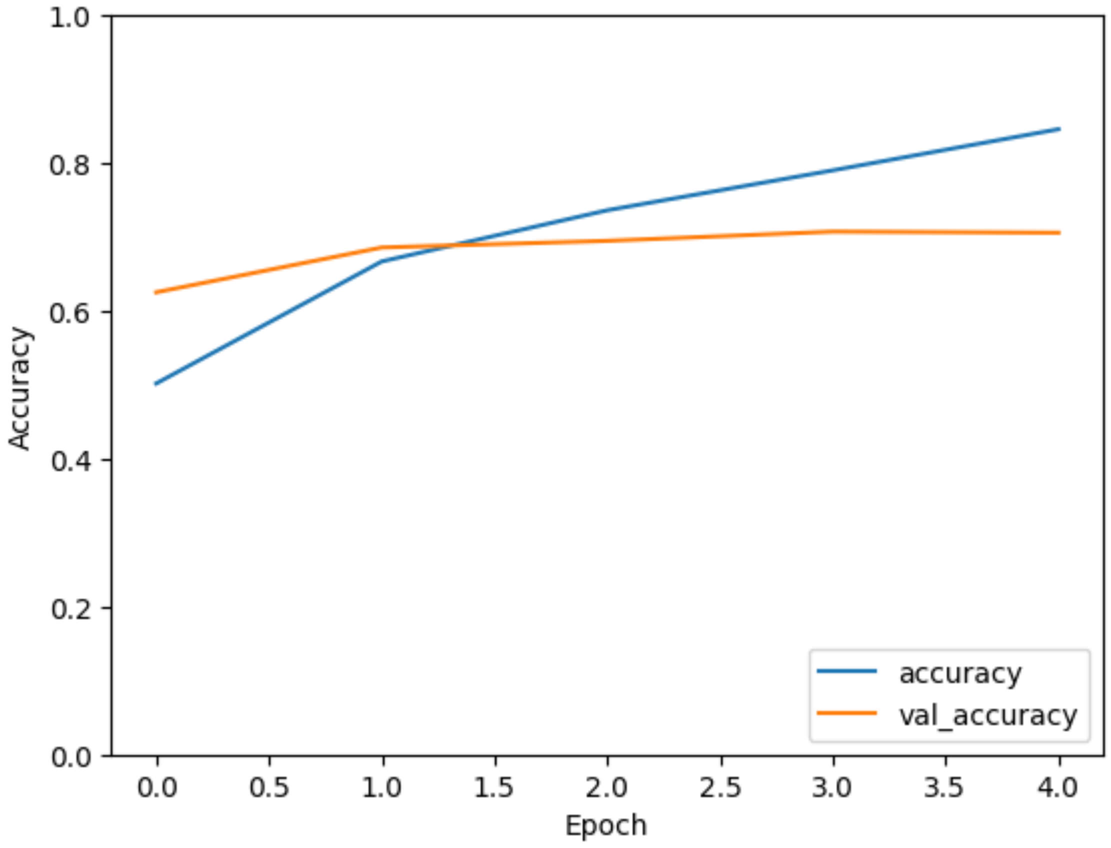



```
-----------------------------------
```

Using 3 kernel size and 48 channels

```
cnn_build(3,48)
```

```
Epoch 1/5
1563/1563 [==============================] - 19s 11ms/step - loss: 1.3856 - accuracy: 0.5018 - val_loss: 1.0578 - val_accuracy: 0.6252
Epoch 2/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.9465 - accuracy: 0.6669 - val_loss: 0.9083 - val_accuracy: 0.6858
Epoch 3/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.7563 - accuracy: 0.7362 - val_loss: 0.8792 - val_accuracy: 0.6948
Epoch 4/5
1563/1563 [==============================] - 16s 11ms/step - loss: 0.5969 - accuracy: 0.7902 - val_loss: 0.8915 - val_accuracy: 0.7073
Epoch 5/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.4403 - accuracy: 0.8458 - val_loss: 0.9679 - val_accuracy: 0.7058
-----------------------------------
313/313 - 1s - loss: 0.9679 - accuracy: 0.7058 - 1s/epoch - 4ms/step
Test accuracy: 0.7057999968528748
-----------------------------------
```




```
-----------------------------------
```

Among all 3 Conv Layer with **(33) kernel size and 48 channel performed best***

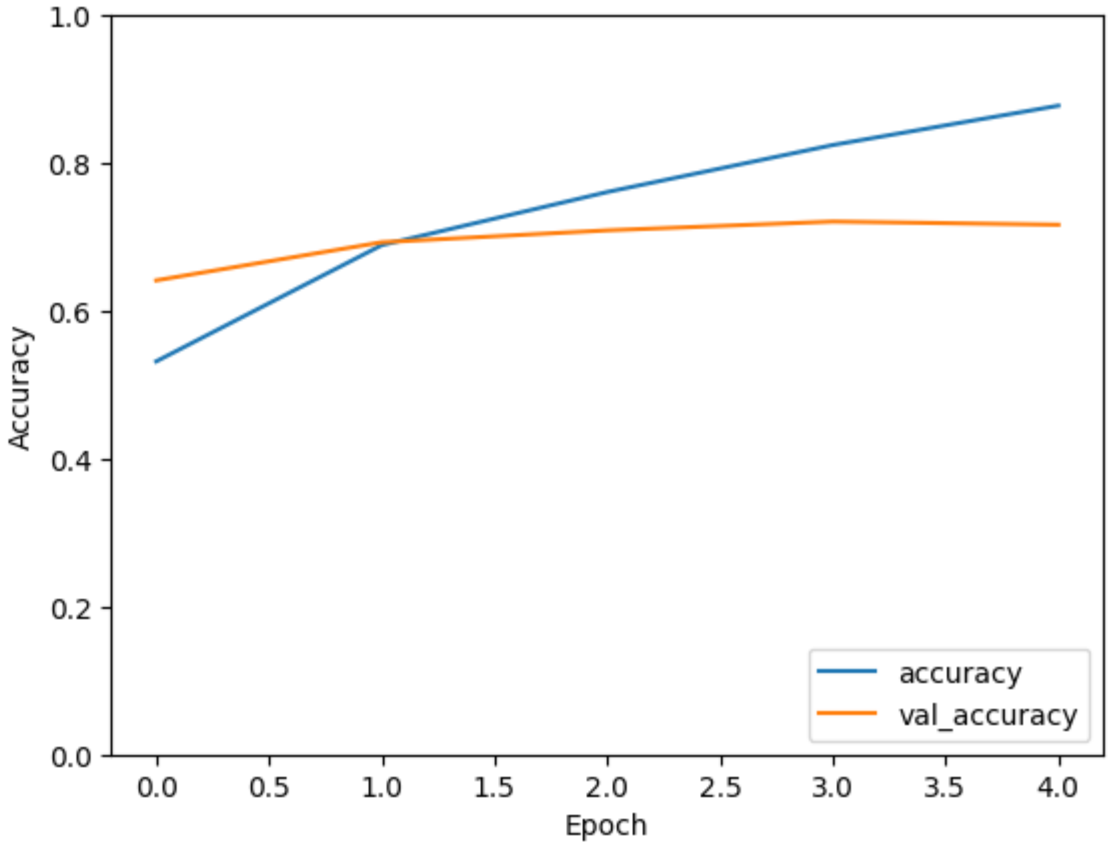
3) Implement the CNN model with different type of pooling method and discuss the interpretation happen due to these changes.

Using AveragePooling

```
# Define the CNN model
model = models.Sequential()

# Convolutional layers with varying parameters
model.add(layers.Conv2D(48, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(48*2, (3, 3), activation='relu'))
model.add(layers.Conv2D(48*4, (3, 3), activation='relu'))
model.add(layers.AveragePooling2D((2, 2)))

# Flatten and add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))

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

# Train the model
history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
print("-----------------------------------")
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")
print("-----------------------------------")
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
print()
print("-----------------------------------")
print()
```

```
Epoch 1/5
1563/1563 [==============================] - 19s 11ms/step - loss: 1.3088 - accuracy: 0.5317 - val_loss: 1.0279 - val_accuracy: 0.6413
Epoch 2/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.8853 - accuracy: 0.6890 - val_loss: 0.9018 - val_accuracy: 0.6929
Epoch 3/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.6866 - accuracy: 0.7609 - val_loss: 0.8482 - val_accuracy: 0.7090
Epoch 4/5
1563/1563 [==============================] - 16s 11ms/step - loss: 0.5059 - accuracy: 0.8245 - val_loss: 0.8610 - val_accuracy: 0.7209
Epoch 5/5
1563/1563 [==============================] - 17s 11ms/step - loss: 0.3485 - accuracy: 0.8777 - val_loss: 1.0453 - val_accuracy: 0.7166
-----------------------------------
313/313 - 1s - loss: 1.0453 - accuracy: 0.7166 - 1s/epoch - 3ms/step
Test accuracy: 0.7166000008583069
-----------------------------------
```



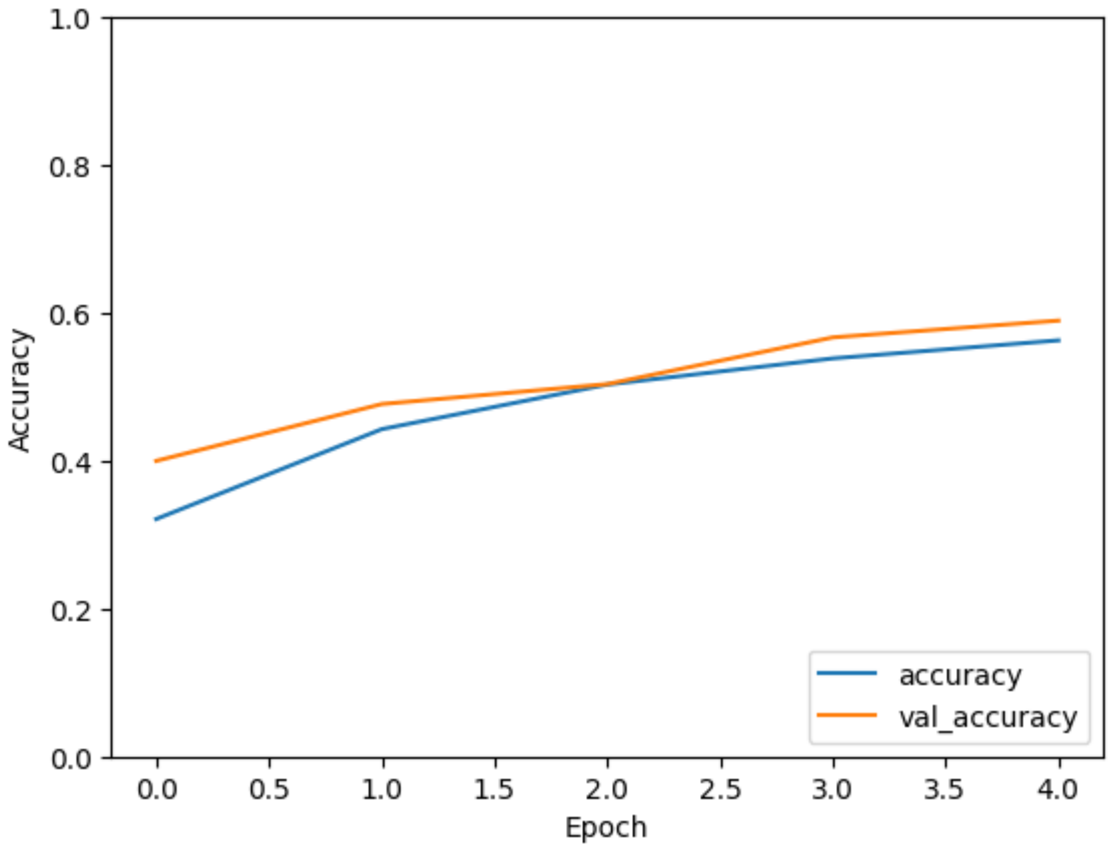

```
-----------------------------------
```

Using GlobalAveragePooling

```
# Define the CNN model
model = models.Sequential()

# Convolutional layers with varying parameters
model.add(layers.Conv2D(48, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(48*2, (3, 3), activation='relu'))
model.add(layers.Conv2D(48*4, (3, 3), activation='relu'))
model.add(layers.GlobalAveragePooling2D())

# Flatten and add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))

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

# Train the model
history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
print("-----------------------------------")
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")
print("-----------------------------------")

# Plot training history
```

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
print()
print("-----------------------------------")
print()
```

```
Epoch 1/5
1563/1563 [==============================] - 19s 11ms/step - loss: 1.7723 - accuracy: 0.3211 - val_loss: 1.5761 - val_accuracy: 0.3998
Epoch 2/5
1563/1563 [==============================] - 16s 10ms/step - loss: 1.5014 - accuracy: 0.4427 - val_loss: 1.4114 - val_accuracy: 0.4767
Epoch 3/5
1563/1563 [==============================] - 16s 10ms/step - loss: 1.3702 - accuracy: 0.5031 - val_loss: 1.3596 - val_accuracy: 0.5037
Epoch 4/5
1563/1563 [==============================] - 16s 10ms/step - loss: 1.2836 - accuracy: 0.5385 - val_loss: 1.2076 - val_accuracy: 0.5669
Epoch 5/5
1563/1563 [==============================] - 16s 10ms/step - loss: 1.2139 - accuracy: 0.5629 - val_loss: 1.1538 - val_accuracy: 0.5896
-----------------------------------
313/313 - 1s - loss: 1.1538 - accuracy: 0.5896 - 1s/epoch - 3ms/step
Test accuracy: 0.5896000266075134
-----------------------------------
```




```
-----------------------------------
```

Among all the Pooling Layer, 'AveragePooling' perfomed best

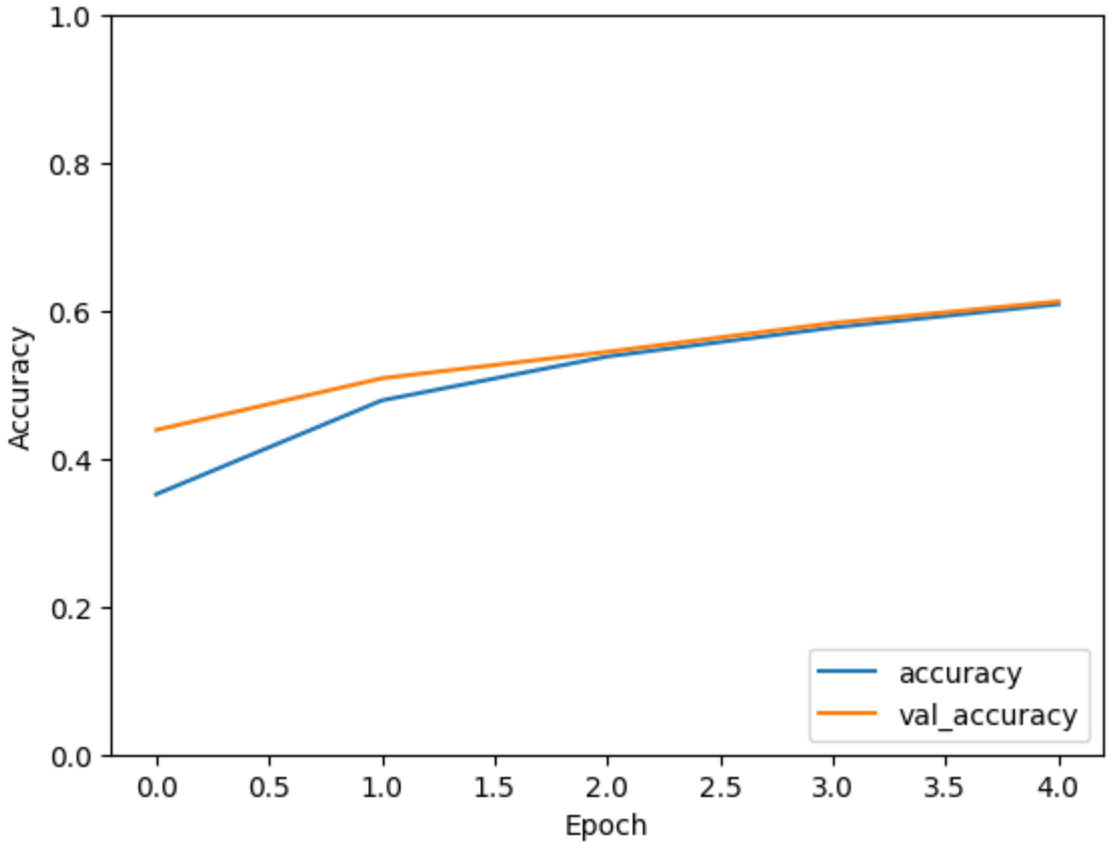
4) Apply different number of Stride and Padding techniques used for good result in CNN.

Including Strides and Padding

```
# Define the CNN model
model = models.Sequential()

# Convolutional layers with varying parameters
model.add(layers.Conv2D(48, (3, 3), activation='relu',padding='same', strides=(2, 2), input_shape=(32, 32, 3)))
model.add(layers.Conv2D(48*2, (3, 3),padding='same', strides=(2, 2), activation='relu'))
model.add(layers.Conv2D(48*4, (3, 3),padding='same', strides=(2, 2), activation='relu'))
model.add(layers.GlobalAveragePooling2D())

# Flatten and add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))

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

# Train the model
history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
print("-----------------------------------")
```

```
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")
print("-----------------------------------")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
print()
print("-----------------------------------")
print()
```

```
Epoch 1/5
1563/1563 [==============================] - 12s 6ms/step - loss: 1.7360 - accuracy: 0.3520 - val_loss: 1.5033 - val_accuracy: 0.4390
Epoch 2/5
1563/1563 [==============================] - 9s 5ms/step - loss: 1.4207 - accuracy: 0.4789 - val_loss: 1.3356 - val_accuracy: 0.5089
Epoch 3/5
1563/1563 [==============================] - 9s 5ms/step - loss: 1.2681 - accuracy: 0.5384 - val_loss: 1.2497 - val_accuracy: 0.5447
Epoch 4/5
1563/1563 [==============================] - 11s 7ms/step - loss: 1.1666 - accuracy: 0.5774 - val_loss: 1.1676 - val_accuracy: 0.5836
Epoch 5/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0848 - accuracy: 0.6091 - val_loss: 1.0811 - val_accuracy: 0.6128
-----------------------------------
313/313 - 1s - loss: 1.0811 - accuracy: 0.6128 - 719ms/epoch - 2ms/step
Test accuracy: 0.6128000020980835
-----------------------------------
```

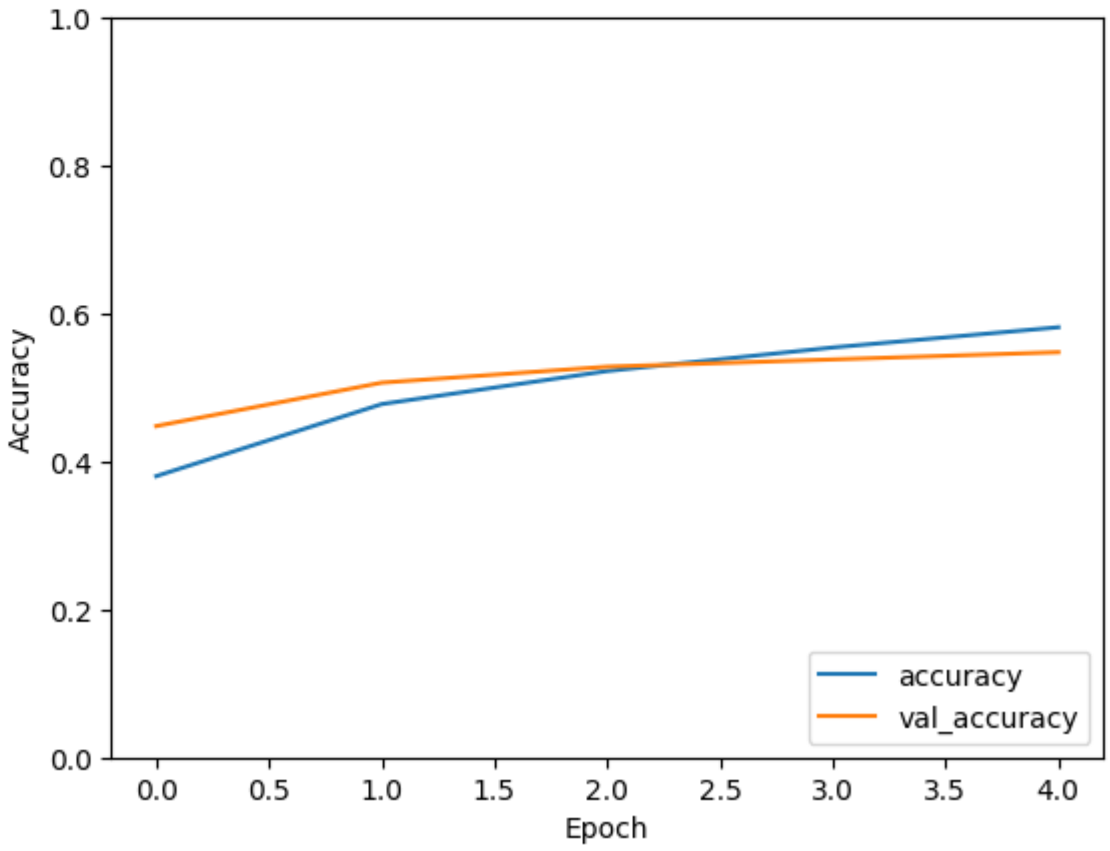



```
-----------------------------------
```

```
# Define the CNN model
model = models.Sequential()

# Convolutional layers with varying parameters
model.add(layers.Conv2D(48, (3, 3), activation='relu',padding='same', strides=(4, 4), input_shape=(32, 32, 3)))
model.add(layers.Conv2D(48*2, (3, 3),padding='same', strides=(4, 4), activation='relu'))
model.add(layers.Conv2D(48*4, (3, 3),padding='same', strides=(4, 4), activation='relu'))
model.add(layers.GlobalAveragePooling2D())

# Flatten and add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))

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

# Train the model
history = model.fit(train_images, train_labels, epochs=5,
                    validation_data=(test_images, test_labels))
print("-----------------------------------")
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")
print("-----------------------------------")
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
print()
print("-----------------------------------")
print()
```

```
Epoch 1/5
1563/1563 [==============================] - 11s 6ms/step - loss: 1.7024 - accuracy: 0.3802 - val_loss: 1.5150 - val_accuracy: 0.4479
Epoch 2/5
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4434 - accuracy: 0.4776 - val_loss: 1.3705 - val_accuracy: 0.5066
Epoch 3/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3246 - accuracy: 0.5223 - val_loss: 1.2988 - val_accuracy: 0.5282
Epoch 4/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.2357 - accuracy: 0.5542 - val_loss: 1.2773 - val_accuracy: 0.5380
Epoch 5/5
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1653 - accuracy: 0.5815 - val_loss: 1.2489 - val_accuracy: 0.5478
-----------------------------------
313/313 - 1s - loss: 1.2489 - accuracy: 0.5478 - 946ms/epoch - 3ms/step
Test accuracy: 0.5478000044822693
-----------------------------------
```




```
-----------------------------------
```

Padding and incresing Stride decreses Accuracy, thus model is not able to capture patterns

5) Adjust the CNN model with the various hyper parameter tuning (Optimizer, Activation function, epoch, Learning rate etc.)

Instead of Adam using Adagrad Optimisers with 3 epoch

```
from tensorflow.keras.optimizers import Adagrad
```

```
def build_cnn(activation):
    model = Sequential([
        Conv2D(32, (3, 3), activation=activation, input_shape=(32, 32, 3)),
        Conv2D(64, (3, 3), activation=activation),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(64, activation=activation),
        Dense(10, activation='softmax')
    ])
    return model
data = []
```

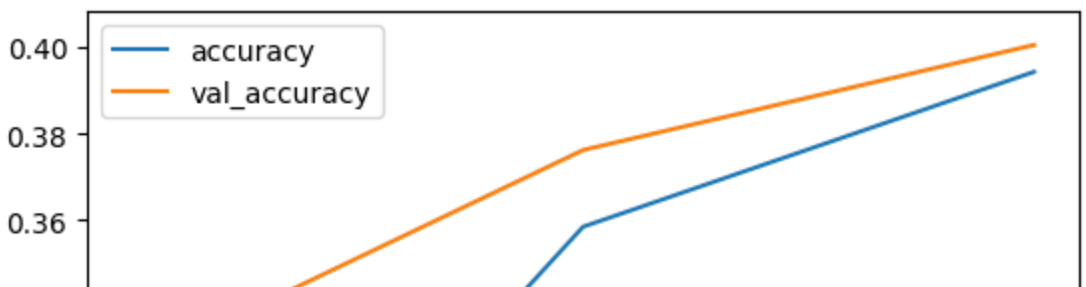
```
from sklearn.metrics import confusion_matrix, recall_score, f1_score, precision_score, classification_report

def train_and_evaluate(learning_rate, activation):
    model = build_cnn(activation)
    model.compile(optimizer=Adagrad(learning_rate=learning_rate),
```

```
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(train_images, train_labels, epochs=3, validation_split=0.2)
    l =[]
    # Evaluate the model
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    y_pred = np.argmax(model.predict(test_images), axis=-1)

    print("Test accuracy:", test_acc)
    print("Confusion matrix:\n", confusion_matrix(test_labels, y_pred))
    print("Classification report:\n", classification_report(test_labels, y_pred))
    precision = precision_score(test_labels, y_pred, average='weighted')
    recall = recall_score(test_labels, y_pred, average='weighted')
    f1 = f1_score(test_labels, y_pred, average='weighted')
    l = [learning_rate,activation,test_acc,confusion_matrix(test_labels, y_pred),classification_report(test_labels, y_pred),precision,reca
    data.append(l)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    # Plot learning curves
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

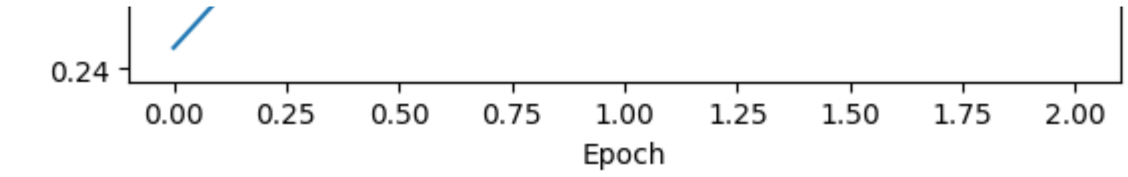
```
learning_rates = [0.001, 0.01, 0.1]
activation_functions = [relu, sigmoid, tanh]
```

```
for lr in learning_rates:
    for activation in activation_functions:
        print(f"Learning rate: {lr}, Activation: {activation.__name__}")
        train_and_evaluate(lr, activation)
        print("-------------------------------")
```

```
Learning rate: 0.001, Activation: relu
Epoch 1/3
1250/1250 [==============================] - 8s 5ms/step - loss: 2.1108 - accuracy: 0.2448 - val_loss: 1.9362 - val_accuracy: 0.3263
Epoch 2/3
1250/1250 [==============================] - 5s 4ms/step - loss: 1.8493 - accuracy: 0.3584 - val_loss: 1.7954 - val_accuracy: 0.3762
Epoch 3/3
1250/1250 [==============================] - 7s 6ms/step - loss: 1.7454 - accuracy: 0.3943 - val_loss: 1.7215 - val_accuracy: 0.4005
313/313 [==============================] - 1s 3ms/step - loss: 1.7091 - accuracy: 0.4116
313/313 [==============================] - 1s 2ms/step
Test accuracy: 0.4115999937057495
Confusion matrix:
 [[486  77  29  45  27  31  24  34 211  36]
 [ 49 641   2  35   5  31  58  29  62  88]
 [112  58 120 117 242 131  96  62  45  17]
 [ 31  93  32 332  59 217 111  59  33  33]
 [ 51  38  43 110 383 106 138  90  31  10]
 [ 19  53  47 200  82 398  70  79  38  14]
 [  5  56  33 186 131  80 446  21  14  28]
 [ 40  68  13 141  85 116  41 409  34  53]
 [139 121   7  36   4  44   7  22 555  65]
 [ 55 322   4  46   7  14  56  41 109 346]]
Classification report:
               precision    recall  f1-score   support

           0       0.49      0.49      0.49      1000
           1       0.42      0.64      0.51      1000
           2       0.36      0.12      0.18      1000
           3       0.27      0.33      0.30      1000
           4       0.37      0.38      0.38      1000
           5       0.34      0.40      0.37      1000
           6       0.43      0.45      0.44      1000
           7       0.48      0.41      0.44      1000
           8       0.49      0.56      0.52      1000
           9       0.50      0.35      0.41      1000

    accuracy                           0.41     10000
   macro avg       0.42      0.41      0.40     10000
weighted avg       0.42      0.41      0.40     10000

Precision: 0.415741500800354
Recall: 0.4116
F1-score: 0.4026740934175066
```



Analysing all the results, we can conclude that ADAM optimser with RELU as activation function having 33* kernel size and 48 channels performed best among all the combinations.




```
--------------------------------
Learning rate: 0.001, Activation: sigmoid
Epoch 1/3
1250/1250 [==============================] - 9s 5ms/step - loss: 2.3075 - accuracy: 0.1036 - val_loss: 2.3052 - val_accuracy: 0.1003
Epoch 2/3
1250/1250 [==============================] - 6s 5ms/step - loss: 2.3047 - accuracy: 0.1009 - val_loss: 2.3032 - val_accuracy: 0.1016
Epoch 3/3
1250/1250 [==============================] - 6s 5ms/step - loss: 2.3039 - accuracy: 0.1027 - val_loss: 2.3037 - val_accuracy: 0.1014
313/313 [==============================] - 1s 3ms/step - loss: 2.3035 - accuracy: 0.1000
313/313 [==============================] - 1s 2ms/step
Test accuracy: 0.10000000149011612
Confusion matrix:
 [[   0 1000    0    0    0    0    0    0    0    0]
 [   0 1000    0    0    0    0    0    0    0    0]
 [   0 1000    0    0    0    0    0    0    0    0]
 [   0 1000    0    0    0    0    0    0    0    0]
 [   0 1000    0    0    0    0    0    0    0    0]
 [   0 1000    0    0    0    0    0    0    0    0]
 [   0 1000    0    0    0    0    0    0    0    0]
```