University of East London

Ain Shams University

Faculty of Engineering

AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

CREDIT HOURS ENG. PROGRAM

Computer engineering and software systems

Fall 24 || Semester 1

Project

Lead Classification using CNN

Deep learning

Submitted to:

Prof. Mahmoud Khalil

Eng. Mahmoud Soheil

Submitted by:

Mohamed Hesham El Said Zidan	20P7579
Seif eldin Ashraf Mahmoud Aref	20P7101
Seif eldin Amr Mostafa Yassine	20p2006

Introduction

In this project we used convolution neural networks to classify leaf images to their correct species and also using the numeric features present in the CSV file to classify the leafs

Image classification

Functions used

```
def read_images(source_path,imgs):
    """
    This function takes a path of images and read them using opencv.

    It returns a list of images.
    """
    for item in os.listdir(source_path):
        item_path = os.path.join(source_path, item)

        if os.path.isfile(item_path):
            # Process the file here
            img  = cv2.imread(item_path)
            imgs.append(img)
        elif os.path.isdir(item_path):
            # Recursively process the subdirectory
            read_images(item_path,imgs)
    return imgs
```

Used for reading the images folder and returning a list of images

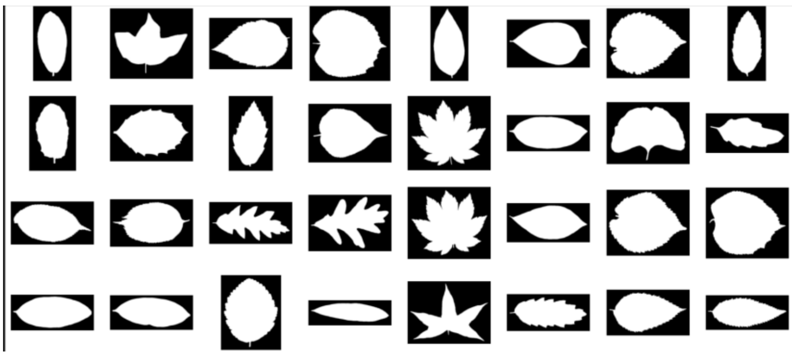
```
def data_generator():
    datagen = ImageDataGenerator(
        rotation_range=40,
        # width_shift_range=0.2,
        # height_shift_range=0.2,
        # shear_range=0.2,
        # zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
    return datagen
```

Used for generating extra images by taking an image and adding to it rotation in the range of 40 degrees and also applying horizontal flip

```python
# Making Directory for the labeled data only
Labeled_path = "/content/Unzipped/Labled_images"

os.makedirs(Labeled_path, exist_ok=True)

# looping on each specie in the species from the train.csv
for specie in list(train["species"].unique()):

  # we are making a Directory inside Labeled for every specie with its name
  specie_path = os.path.join(Labeled_path,specie)
  os.makedirs(specie_path, exist_ok=True)

  # getting all the IDs of the leafs with the specific specie we are in and storing it in a list
  filtered_specie = train[train["species"] == specie]
  specie_ids = list(filtered_specie["id"])

  # looping on all IDs we got for that specific specie and getting its path from the "Unlableled" directory
  for id in specie_ids:
    source_file_path = os.path.join(UnLabled_path, str(id)+".jpg")
    destination_file_path = os.path.join(specie_path, str(id)+".jpg")

    # Moving the image with matching ID from the "Unlableled" directory to "Lableled" directory inside it's specie directory
    shutil.move(source_file_path, destination_file_path)
```

Dividing the data of the images into labeled and unlabeled to divide the train data and the test data then dividing the train data of each class in a separate folder\

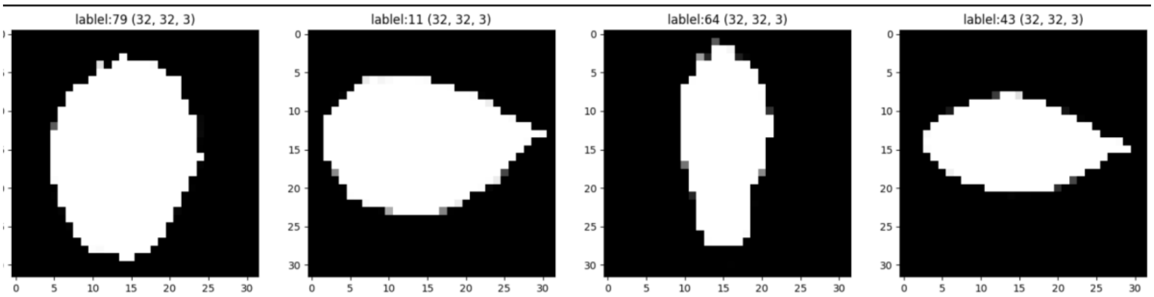
```python
plt.figure(figsize=(18, 8))

num_rows = 4
num_cols = 8

# plot the first 32 images in the images folder
for i in range(num_rows * num_cols):
    ax = plt.subplot(num_rows, num_cols, i + 1)
    plt.imshow(imgs[i])
    plt.axis("off")
```



After reading the images visualize the first 32 images

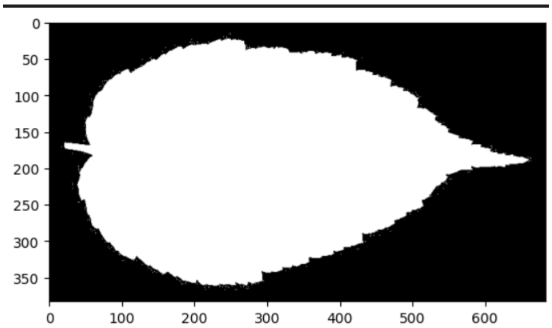
```python
data = tf.keras.utils.image_dataset_from_directory(Labeled_path,image_size=(32, 32),color_mode="rgb",pad_to_aspect_ratio=True)
data
```

```python
data_iterator = data.as_numpy_iterator()
batch = data_iterator.next()
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(f"lablel:{batch[1][idx]} {img.shape}")
```



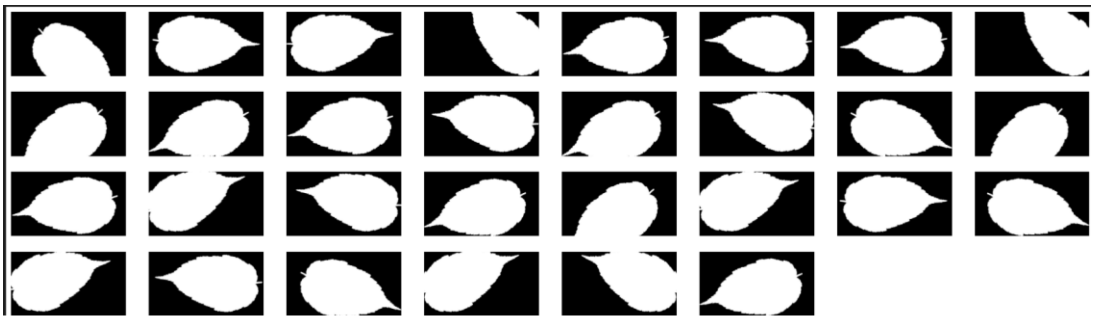
After reading the images used tensor flow library to do preprocessing by changing the sizes of all images to the same size and maintaining the aspect ratio of the image

```
datagen = data_generator()

img = load_img(f"/content/Unzipped/Labled_images/Acer_Capillipes/"+os.listdir("/content/Unzipped/Labled_images/Acer_Capillipes")[0])
x = img_to_array(img)
plt.imshow(x)
x = np.expand_dims(x, axis=0)

aug_path  = "/content/augmented_images"
os.makedirs(aug_path, exist_ok=True)

# Generate 30 variations of the orginal image
i = 0
for batch in datagen.flow(x, batch_size=1, save_to_dir='augmented_images', save_prefix='augmented', save_format='jpg'):
    i += 1
    if i >= 30:
        break
```



Original image

```
plt.figure(figsize=(28, 8))

num_rows = 4
num_cols = 8

# plot the first 32 images in the array 32x32 each.
for i in range(len(os.listdir(aug_path))):
    ax = plt.subplot(num_rows, num_cols, i + 1)
    plt.imshow(augmented[i])
    plt.axis("off")
```



After augmentation

Generate data that is augmented and visualize it

CNN Model

```python
@dataclass(frozen=True)
class DatasetConfig:
    NUM_CLASSES:  int = 99
    IMG_HEIGHT:   int = 256
    IMG_WIDTH:    int = 256
    NUM_CHANNELS: int = 1

@dataclass(frozen=True)
class TrainingConfig:
    EPOCHS: int = 30
    BATCH_SIZE: int = 32
    LEARNING_RATE: float = 0.001
    OPTIMIZERS = [keras.optimizers.Adam(learning_rate=LEARNING_RATE), keras.optimizers.RMSprop(learning_rate=LEARNING_RATE), keras.optimizers.SGD(learning_rate=LEARNING_RATE)]
```

Classes that have a pre configurations for some parameters of the model

```python
# make new Directory for the Augmented Data
os.makedirs("Augmneted_Labled", exist_ok=True)

# Augment Data from "Labeled" Data Directory
Data_augmentation(Labeled_path, "Augmneted_Labled", data_generator())
```

```python
len(os.listdir("/content/Augmneted_Labled")) == 99
```

```python
# converting the "Augmneted_Labled" Directory to Zip file
shutil.make_archive("/content/Augmneted_Labled", 'zip', 'Augmneted_Labled')
# moving the Zip file to mydrive
shutil.move("/content/Augmneted_Labled.zip", "/content/drive/MyDrive/ZIPs/Augmneted_Labled.zip")
```

```
ntent/drive/MyDrive/ZIPs/Augmneted_Labled.zip'
```

```python
path_in_drive = "/content/drive/MyDrive/ZIPs/Augmneted_Labled.zip"

# unziping the Augmneted_Labled Zip folder in my local directory "Augmneted_Labled"
!unzip "/content/drive/MyDrive/ZIPs/Augmneted_Labled.zip" -d "/content/Augmneted_Labled"
```

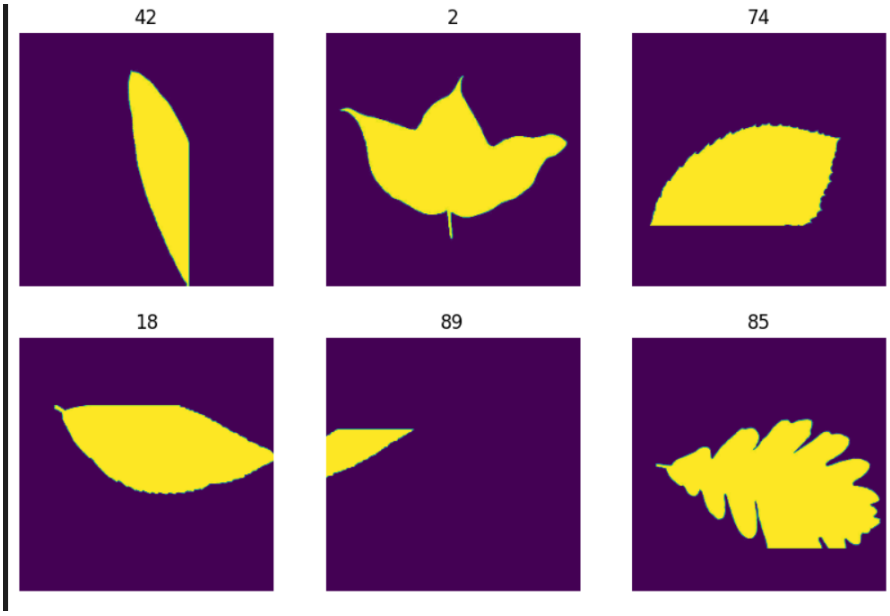
Adding the original images with the augmented images folder

```python
train_ds, test_ds = tf.keras.utils.image_dataset_from_directory("/content/Augmneted_Labled",
                                                                validation_split=0.2,
                                                                subset="both",
                                                                image_size=(DatasetConfig.IMG_HEIGHT, DatasetConfig.IMG_WIDTH),
                                                                batch_size=TrainingConfig.BATCH_SIZE,
                                                                seed = 124,
                                                                color_mode="grayscale",
                                                                pad_to_aspect_ratio=True
                                                                )
```

```
Found 30269 files belonging to 99 classes.
Using 24216 files for training.
Using 6053 files for validation.
```

Final preprocessing on the images are done to resize them to 256 as 32 was too small and removed a lot of features and data augmentation increased the number of train data to 30269

```python
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(np.array(images[i]))
        plt.title(int(labels[i]))
        plt.axis("off")
```



Visualization of the result of the data after all the preprocessing

CNN Model generation

```python
def CNN_block(model, No_of_conv_Layers,filters_no, Dropout_rate):

  # adding all the 2D conv_layers
  for i in range(No_of_conv_Layers):
    model.add(Conv2D(filters=filters_no, kernel_size=3, padding='same', activation='relu'))

  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(Dropout_rate))
```

Function used to generate block of CNN

```python
def training(No_of_conv_Layers, No_of_CNN_blocks, Dropout_rate, optimizer, Learning_rate, train_ds, val_ds, model_name, weight_decay=None):

  model = Sequential()

  model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='relu', input_shape=(DatasetConfig.IMG_WIDTH, DatasetConfig.IMG_HEIGHT, DatasetConfig.NUM_CHANNELS)))

  # intial block is always 32 filters
  CNN_block(model, No_of_conv_Layers-1,32,Dropout_rate)

  for i in range(No_of_CNN_blocks-1):
    CNN_block(model, No_of_conv_Layers,64,Dropout_rate)

  # Final neural network block
  model.add(Flatten())
  model.add(Dense(512, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(DatasetConfig.NUM_CLASSES, activation='softmax'))

  model.summary()

  while(True):
    response = input("Are you satisfied by the model architecture above ? [Y/N] :")
    if response.lower() == "y":

      response = input("Train or Return the Model Archetecture ? [T/M] :")
      if response.lower() == "t":
        break
      elif response.lower() == "m":
        return model

    elif response.lower() == "n":
      return
```

```python
    else:
        continue

  # we used sparsedCategorigal loss because the true labels are integers
  # we use CategorgicalCrossEntopy when the labels are one-hot encoded
  model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=["accuracy"],
  )

  os.makedirs(os.path.join('Logs'), exist_ok=True)
  log_save_path_in_Colab = "/content/Logs"
  weights_save_path_in_Drive = f'/content/drive/MyDrive/Models/LeafClassification_model_{str(model_name)}.weights.h5'
  tensorboard_callback = TensorBoard(log_dir=log_save_path_in_Colab)
  checkpoint_callback = ModelCheckpoint(
      filepath = weights_save_path_in_Drive,
      save_weights_only=True,
      monitor='val_loss',
      mode='min',
      save_best_only=True
  )

# batch size is already specificed in the train_ds and val_ds by tensorflow
  model.fit(
      train_ds,
      # epochs=TrainingConfig.EPOCHS,
      epochs=TrainingConfig.EPOCHS,
      validation_data=val_ds,
      callbacks=[tensorboard_callback,checkpoint_callback]
  )
  # saving logs in Drive
  log_save_path_in_Drive = '/content/drive/MyDrive/Models/Logs/Log_LeafClassification_model_'+str(model_name)
```

```python
  log_save_path_in_Drive = '/content/drive/MyDrive/Models/Logs/Log_LeafClassification_model_'+str(model_name)
  shutil.copytree(log_save_path_in_Colab, log_save_path_in_Drive)
  # show tensorboard Results
  %tensorboard --logdir Logs

  return model
```

This is the training function that take parameters that takes the number of convolution layers in each block, take the number of convolution blocks which are the number of repeated convolution layers and maxpooling and dropout in each block, takes the drop rate which is a number between 0 and 1 then gets they type of optimizer and learning rate and the train and test datasets, model name and if there is weight decay or not.

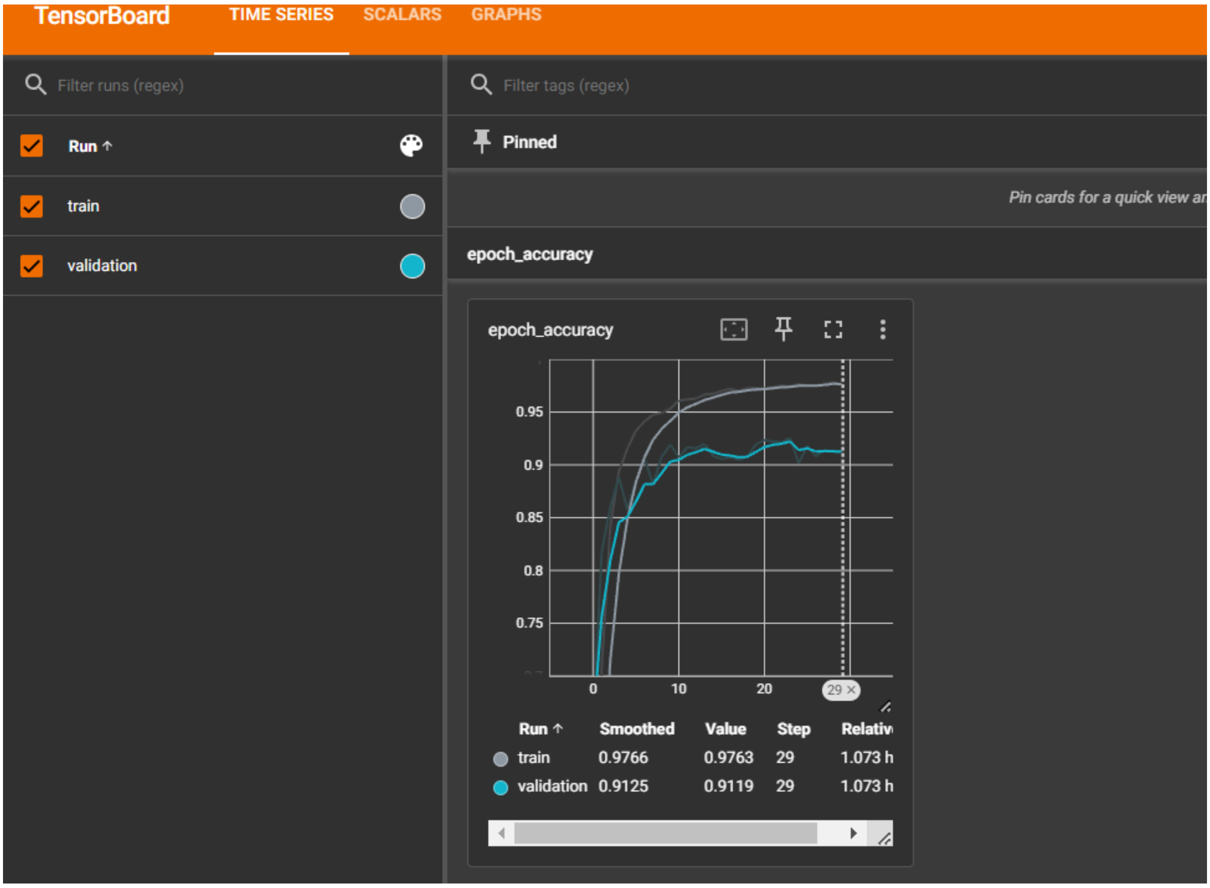
Optimizer comparison

RMS

```python
RMS_model = training(2, 3, 0.25, TrainingConfig.OPTIMIZERS[1], TrainingConfig.LEARNING_RATE, train_ds, test_ds,"RMS_optimizer")
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 256, 256, 32)	320
conv2d_7 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d_3 (MaxPooling2D)	(None, 128, 128, 32)	0
dropout_4 (Dropout)	(None, 128, 128, 32)	0
conv2d_8 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_9 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_5 (Dropout)	(None, 64, 64, 64)	0
conv2d_10 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_11 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_6 (Dropout)	(None, 32, 32, 64)	0
flatten_1 (Flatten)	(None, 65536)	0
dense_2 (Dense)	(None, 512)	33,554,944
dropout_7 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)



Epoch 1/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 126s 161ms/step - accuracy: 0.2336 - loss: 4.6771 - val_accuracy: 0.6549 - val_loss: 1.1194

Epoch 2/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 132s 151ms/step - accuracy: 0.6512 - loss: 1.1198 - val_accuracy: 0.8181 - val_loss: 0.5607

Epoch 3/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 144s 153ms/step - accuracy: 0.8219 - loss: 0.5615 - val_accuracy: 0.8601 - val_loss: 0.4250

Epoch 4/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 114s 151ms/step - accuracy: 0.8867 - loss: 0.3595 - val_accuracy: 0.8880 - val_loss: 0.3722

Epoch 5/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 111s 147ms/step - accuracy: 0.9121 - loss: 0.2735 - val_accuracy: 0.8579 - val_loss: 0.4970

Epoch 6/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 147ms/step - accuracy: 0.9304 - loss: 0.2302 - val_accuracy: 0.8852 - val_loss: 0.3760

Epoch 7/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 112s 148ms/step - accuracy: 0.9400 - loss: 0.2017 - val_accuracy: 0.9052 - val_loss: 0.3622

Epoch 8/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 138s 143ms/step - accuracy: 0.9465 - loss: 0.1889 - val_accuracy: 0.8819 - val_loss: 0.3894

Epoch 9/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 145s 147ms/step - accuracy: 0.9481 - loss: 0.1783 - val_accuracy: 0.9078 - val_loss: 0.4349

Epoch 10/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 117s 155ms/step - accuracy: 0.9525 - loss: 0.1670 - val_accuracy: 0.9190 - val_loss: 0.3533

Epoch 11/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 136s 147ms/step - accuracy: 0.9620 - loss: 0.1464 - val_accuracy: 0.9072 - val_loss: 0.4937

Epoch 12/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 107s 142ms/step - accuracy: 0.9610 - loss: 0.1449 - val_accuracy: 0.9169 - val_loss: 0.3617

Epoch 13/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 143s 144ms/step - accuracy: 0.9623 - loss: 0.1392 - val_accuracy: 0.9156 - val_loss: 0.4061

Epoch 14/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 144ms/step - accuracy: 0.9654 - loss: 0.1440 - val_accuracy: 0.9199 - val_loss: 0.4048

Epoch 15/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 145s 147ms/step - accuracy: 0.9679 - loss: 0.1284 - val_accuracy: 0.9085 - val_loss: 0.3542

Epoch 16/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 147ms/step - accuracy: 0.9687 - loss: 0.1372 - val_accuracy: 0.9055 - val_loss: 0.6022

Epoch 17/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 139s 143ms/step - accuracy: 0.9731 - loss: 0.1133 - val_accuracy: 0.9073 - val_loss: 0.4045

Epoch 18/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 141s 142ms/step - accuracy: 0.9688 - loss: 0.1232 - val_accuracy: 0.9048 - val_loss: 0.4015

Epoch 19/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 144s 144ms/step - accuracy: 0.9717 - loss: 0.1361 - val_accuracy: 0.9081 - val_loss: 0.6174

Epoch 20/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 140s 142ms/step - accuracy: 0.9714 - loss: 0.1314 - val_accuracy: 0.9194 - val_loss: 0.4987

Epoch 21/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 146s 148ms/step - accuracy: 0.9734 - loss: 0.1228 - val_accuracy: 0.9237 - val_loss: 0.4835

Epoch 22/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 107s 141ms/step - accuracy: 0.9738 - loss: 0.1294 - val_accuracy: 0.9222 - val_loss: 0.7427

Epoch 23/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 111s 147ms/step - accuracy: 0.9734 - loss: 0.1356 - val_accuracy: 0.9215 - val_loss: 0.5292

Epoch 24/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 143s 148ms/step - accuracy: 0.9754 - loss: 0.1335 - val_accuracy: 0.9255 - val_loss: 0.6518

Epoch 25/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 148ms/step - accuracy: 0.9751 - loss: 0.1236 - val_accuracy: 0.9015 - val_loss: 0.7289

Epoch 26/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 141s 147ms/step - accuracy: 0.9753 - loss: 0.1346 - val_accuracy: 0.9182 - val_loss: 0.7542

Epoch 27/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 140s 144ms/step - accuracy: 0.9766 - loss: 0.1382 - val_accuracy: 0.9080 - val_loss: 0.8447

Epoch 28/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 107s 141ms/step - accuracy: 0.9762 - loss: 0.1366 - val_accuracy: 0.9139 - val_loss: 0.6679

Epoch 29/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 144s 144ms/step - accuracy: 0.9781 - loss: 0.1304 - val_accuracy: 0.9124 - val_loss: 0.6498

Epoch 30/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 140s 142ms/step - accuracy: 0.9761 - loss: 0.1873 - val_accuracy: 0.9119 - val_loss: 0.9307

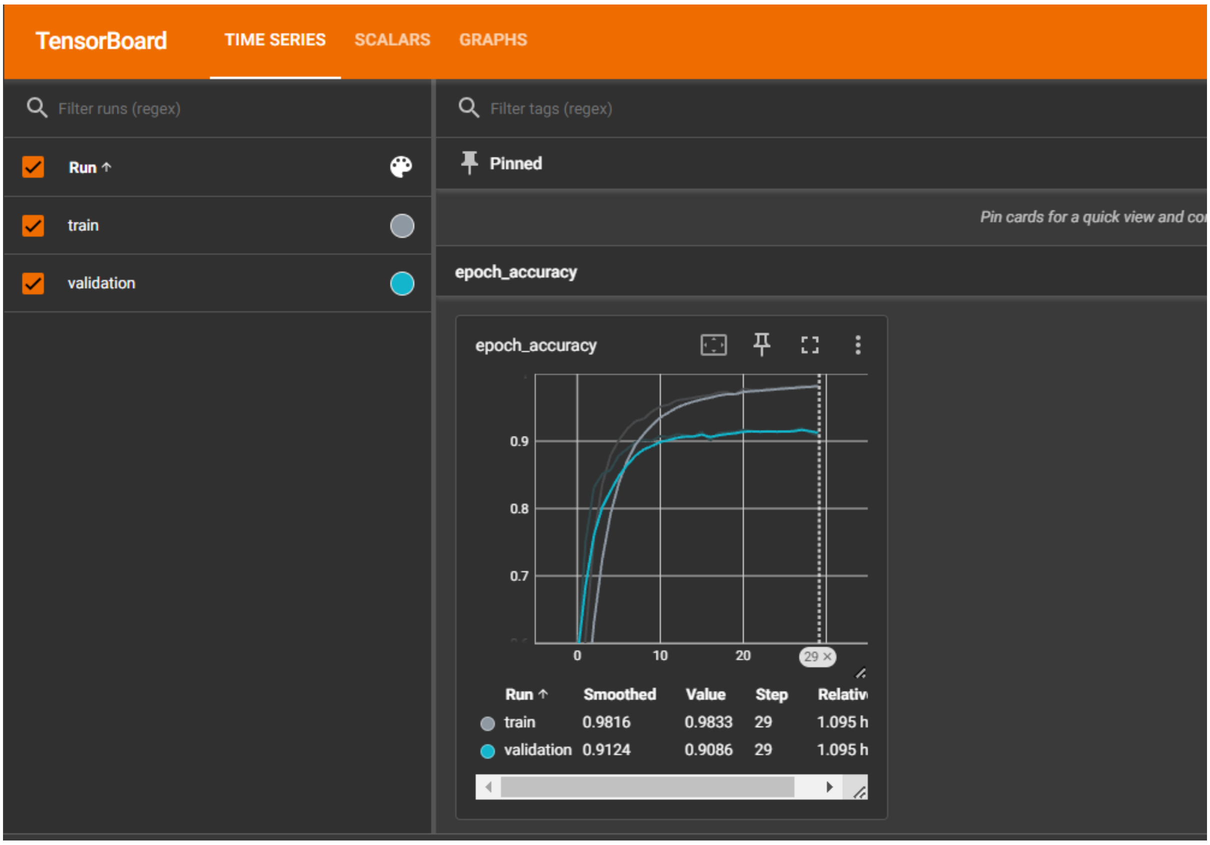
The RMS optimizer has a final train accuracy of 0.9761 and validation accuracy of 0.9119

ADAM optimizer

```
ADAM_model = training(2, 3, 0.25, TrainingConfig.OPTIMIZERS[0], TrainingConfig.LEARNING_RATE, train_ds, test_ds,"ADAM_optimizer")
```

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 256, 256, 32)	320
conv2d_31 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d_15 (MaxPooling2D)	(None, 128, 128, 32)	0
dropout_20 (Dropout)	(None, 128, 128, 32)	0
conv2d_32 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_33 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_16 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_21 (Dropout)	(None, 64, 64, 64)	0
conv2d_34 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_35 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_17 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_22 (Dropout)	(None, 32, 32, 64)	0
flatten_5 (Flatten)	(None, 65536)	0
dense_10 (Dense)	(None, 512)	33,554,944
dropout_23 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)



accuracy: 0.9119 - val_loss: 0.9307

Are you satisfied by the model architecture above ? [Y/N] :y

Train or Return the Model Archetecture ? [T/M] :t

Epoch 1/30

757/757 129s 164ms/step - accuracy: 0.1820 - loss: 4.8390 - val_accuracy: 0.5776 - val_loss: 1.4151

Epoch 2/30

757/757 118s 155ms/step - accuracy: 0.5526 - loss: 1.4809 - val_accuracy: 0.7510 - val_loss: 0.7816

Epoch 3/30

757/757 145s 160ms/step - accuracy: 0.7285 - loss: 0.8506 - val_accuracy: 0.8313 - val_loss: 0.5229

Epoch 4/30

757/757 141s 159ms/step - accuracy: 0.8226 - loss: 0.5506 - val_accuracy: 0.8510 - val_loss: 0.4472

Epoch 5/30

757/757 136s 151ms/step - accuracy: 0.8715 - loss: 0.4004 - val_accuracy: 0.8573 - val_loss: 0.4585

Epoch 6/30

757/757 150s 161ms/step - accuracy: 0.8944 - loss: 0.3262 - val_accuracy: 0.8787 - val_loss: 0.3751

Epoch 7/30

757/757 136s 153ms/step - accuracy: 0.9169 - loss: 0.2606 - val_accuracy: 0.8883 - val_loss: 0.3592

Epoch 8/30

757/757 117s 155ms/step - accuracy: 0.9278 - loss: 0.2272 - val_accuracy: 0.8984 - val_loss: 0.3340

Epoch 9/30

757/757 148s 162ms/step - accuracy: 0.9287 - loss: 0.2122 - val_accuracy: 0.9014 - val_loss: 0.3302

Epoch 10/30

757/757 138s 158ms/step - accuracy: 0.9457 - loss: 0.1695 - val_accuracy: 0.9005 - val_loss: 0.3167

Epoch 11/30

757/757 136s 150ms/step - accuracy: 0.9490 - loss: 0.1577 - val_accuracy: 0.9075 - val_loss: 0.3319

Epoch 12/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 150ms/step - accuracy: 0.9520 - loss: 0.1400 - val_accuracy: 0.9057 - val_loss: 0.3425

Epoch 13/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 149s 160ms/step - accuracy: 0.9602 - loss: 0.1268 - val_accuracy: 0.9101 - val_loss: 0.3071

Epoch 14/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 130s 143ms/step - accuracy: 0.9625 - loss: 0.1206 - val_accuracy: 0.9090 - val_loss: 0.3181

Epoch 15/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 143ms/step - accuracy: 0.9658 - loss: 0.1110 - val_accuracy: 0.9068 - val_loss: 0.3257

Epoch 16/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 146s 149ms/step - accuracy: 0.9664 - loss: 0.1023 - val_accuracy: 0.9143 - val_loss: 0.3090

Epoch 17/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 149ms/step - accuracy: 0.9662 - loss: 0.1043 - val_accuracy: 0.9007 - val_loss: 0.3788

Epoch 18/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 149ms/step - accuracy: 0.9728 - loss: 0.0888 - val_accuracy: 0.9121 - val_loss: 0.3395

Epoch 19/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 149ms/step - accuracy: 0.9736 - loss: 0.0836 - val_accuracy: 0.9131 - val_loss: 0.3132

Epoch 20/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 143s 150ms/step - accuracy: 0.9702 - loss: 0.0934 - val_accuracy: 0.9136 - val_loss: 0.3271

Epoch 21/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 108s 142ms/step - accuracy: 0.9773 - loss: 0.0761 - val_accuracy: 0.9171 - val_loss: 0.3172

Epoch 22/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 147s 149ms/step - accuracy: 0.9762 - loss: 0.0743 - val_accuracy: 0.9156 - val_loss: 0.3448

Epoch 23/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 137s 143ms/step - accuracy: 0.9761 - loss: 0.0782 - val_accuracy: 0.9133 - val_loss: 0.3521

Epoch 24/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 146s 149ms/step - accuracy: 0.9775 - loss: 0.0700 - val_accuracy: 0.9151 - val_loss: 0.3447

Epoch 25/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 112s 148ms/step - accuracy: 0.9778 - loss: 0.0704 - val_accuracy: 0.9134 - val_loss: 0.3263

Epoch 26/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 149ms/step - accuracy: 0.9784 - loss: 0.0661 - val_accuracy: 0.9144 - val_loss: 0.3506

Epoch 27/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 149ms/step - accuracy: 0.9778 - loss: 0.0726 - val_accuracy: 0.9152 - val_loss: 0.3679

Epoch 28/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 139s 144ms/step - accuracy: 0.9818 - loss: 0.0571 - val_accuracy: 0.9197 - val_loss: 0.3091

Epoch 29/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 108s 142ms/step - accuracy: 0.9803 - loss: 0.0621 - val_accuracy: 0.9124 - val_loss: 0.3547

Epoch 30/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 112s 148ms/step - accuracy: 0.9828 - loss: 0.0540 - val_accuracy: 0.9086 - val_loss: 0.3746

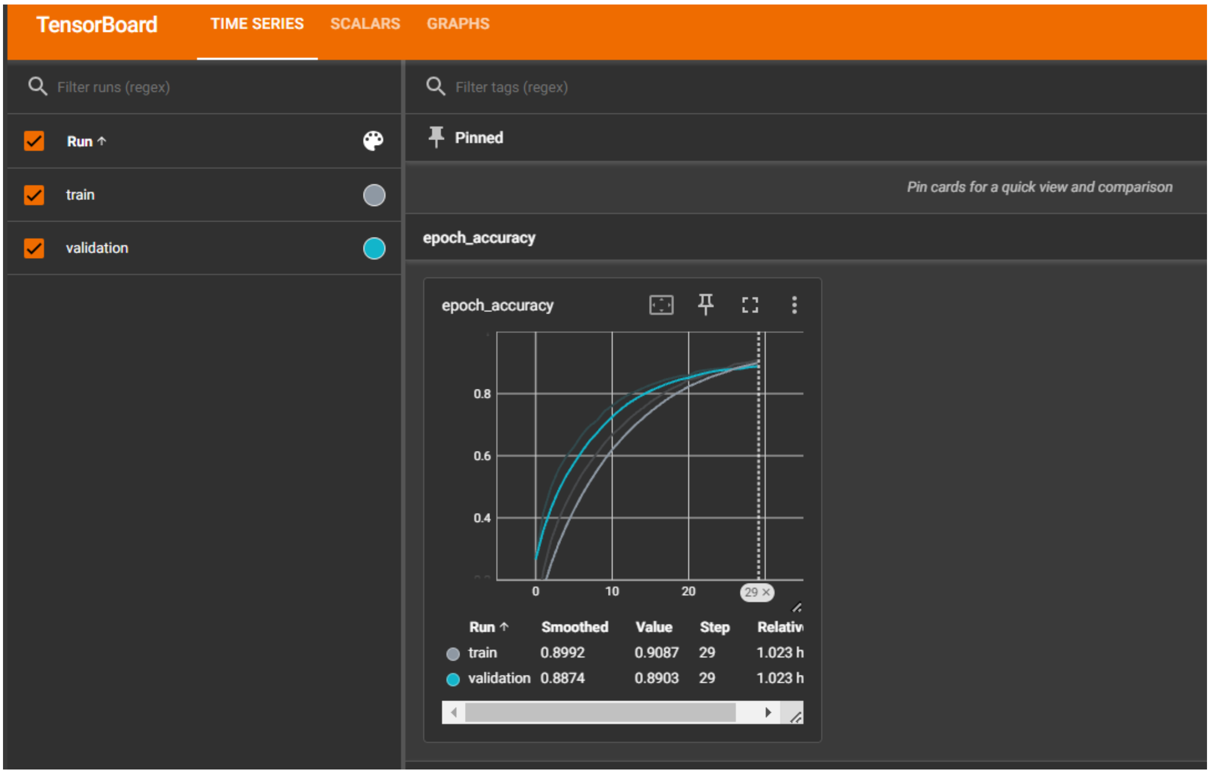
The ADAM optimizer has a final train accuracy of 0.9828 and validation accuracy of 0.9086

SGD

```
SGD_model = training(2, 3, 0.25, TrainingConfig.OPTIMIZERS[2], TrainingConfig.LEARNING_RATE, train_ds, test_ds,"SGD_optimizer")
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 32)	320
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
dropout (Dropout)	(None, 128, 128, 32)	0
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_1 (Dropout)	(None, 64, 64, 64)	0
conv2d_4 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_2 (Dropout)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 512)	33,554,944
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)



```
Epoch 1/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 153s 176ms/step - accuracy: 0.0365 - loss: 5.1151 - val_accuracy: 0.2660 - val_loss: 3.3463
Epoch 2/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 107s 141ms/step - accuracy: 0.1943 - loss: 3.2733 - val_accuracy: 0.4158 - val_loss: 2.4468
Epoch 3/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 105s 139ms/step - accuracy: 0.3024 - loss: 2.6882 - val_accuracy: 0.4983 - val_loss: 2.0690
Epoch 4/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 146s 145ms/step - accuracy: 0.3722 - loss: 2.3315 - val_accuracy: 0.5566 - val_loss: 1.7879
Epoch 5/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 108s 143ms/step - accuracy: 0.4305 - loss: 2.0576 - val_accuracy: 0.5977 - val_loss: 1.5790
Epoch 6/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 154s 158ms/step - accuracy: 0.4848 - loss: 1.8500 - val_accuracy: 0.6288 - val_loss: 1.3895
Epoch 7/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 107s 142ms/step - accuracy: 0.5292 - loss: 1.6552 - val_accuracy: 0.6663 - val_loss: 1.2392
Epoch 8/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 141s 140ms/step - accuracy: 0.5683 - loss: 1.4839 - val_accuracy: 0.6950 - val_loss: 1.1643
Epoch 9/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 110s 145ms/step - accuracy: 0.6026 - loss: 1.3814 - val_accuracy: 0.7129 - val_loss: 1.0312
Epoch 10/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 110s 145ms/step - accuracy: 0.6380 - loss: 1.2384 - val_accuracy: 0.7438 - val_loss: 0.9276
Epoch 11/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 110s 145ms/step - accuracy: 0.6659 - loss: 1.1230 - val_accuracy: 0.7609 - val_loss: 0.8343
Epoch 12/30
757/757 ━━━━━━━━━━━━━━━━━━━━ 139s 141ms/step - accuracy: 0.6824 - loss: 1.0389 - val_accuracy: 0.7798 - val_loss: 0.7855
```

Epoch 13/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 145s 145ms/step - accuracy: 0.7101 - loss: 0.9386 - val_accuracy: 0.7953 - val_loss: 0.7067

Epoch 14/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 140s 143ms/step - accuracy: 0.7261 - loss: 0.8824 - val_accuracy: 0.8064 - val_loss: 0.6651

Epoch 15/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 144s 145ms/step - accuracy: 0.7451 - loss: 0.8046 - val_accuracy: 0.8174 - val_loss: 0.6289

Epoch 16/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 139s 142ms/step - accuracy: 0.7738 - loss: 0.7231 - val_accuracy: 0.8282 - val_loss: 0.5853

Epoch 17/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 142ms/step - accuracy: 0.7870 - loss: 0.6703 - val_accuracy: 0.8364 - val_loss: 0.5630

Epoch 18/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 144s 144ms/step - accuracy: 0.8029 - loss: 0.6200 - val_accuracy: 0.8452 - val_loss: 0.5260

Epoch 19/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 145ms/step - accuracy: 0.8116 - loss: 0.5859 - val_accuracy: 0.8511 - val_loss: 0.5025

Epoch 20/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 145ms/step - accuracy: 0.8265 - loss: 0.5348 - val_accuracy: 0.8578 - val_loss: 0.4774

Epoch 21/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 106s 140ms/step - accuracy: 0.8440 - loss: 0.4851 - val_accuracy: 0.8579 - val_loss: 0.4696

Epoch 22/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 109s 144ms/step - accuracy: 0.8460 - loss: 0.4684 - val_accuracy: 0.8710 - val_loss: 0.4405

Epoch 23/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 140s 141ms/step - accuracy: 0.8563 - loss: 0.4337 - val_accuracy: 0.8749 - val_loss: 0.4341

Epoch 24/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 109s 144ms/step - accuracy: 0.8709 - loss: 0.3967 - val_accuracy: 0.8786 - val_loss: 0.4132

Epoch 25/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 144ms/step - accuracy: 0.8770 - loss: 0.3649 - val_accuracy: 0.8804 - val_loss: 0.4071

Epoch 26/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 105s 138ms/step - accuracy: 0.8816 - loss: 0.3534 - val_accuracy: 0.8815 - val_loss: 0.3957

Epoch 27/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 105s 138ms/step - accuracy: 0.8947 - loss: 0.3219 - val_accuracy: 0.8819 - val_loss: 0.3978

Epoch 28/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 109s 144ms/step - accuracy: 0.8934 - loss: 0.3175 - val_accuracy: 0.8842 - val_loss: 0.3840

Epoch 29/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 142s 144ms/step - accuracy: 0.8978 - loss: 0.3011 - val_accuracy: 0.8920 - val_loss: 0.3757

Epoch 30/30

757/757 ━━━━━━━━━━━━━━━━━━━━ 141s 142ms/step - accuracy: 0.9073 - loss: 0.2771 - val_accuracy: 0.8903 - val_loss: 0.3808

The SGD optimizer has a final train accuracy of 0.9073 and validation accuracy of 0.8903

Final comparison for different optimizers

	Accuaracy	Precision	Recall	F1_Score
SGD_optimizer_train	0.995540	0.995604	0.995540	0.995534
SGD_optimizer_valid	0.891954	0.895749	0.891954	0.892141
ADAM_optimizer_train	0.998844	0.998854	0.998844	0.998844
ADAM_optimizer_valid	0.910127	0.913640	0.910127	0.910109
RMS_optimizer_train	0.997564	0.997612	0.997564	0.997563
RMS_optimizer_valid	0.919048	0.922641	0.919048	0.919354

we can see that RMS optimizer have the best accuracy, precision, Recall, F1_Score between all optimizer with accuracy of 0.919048

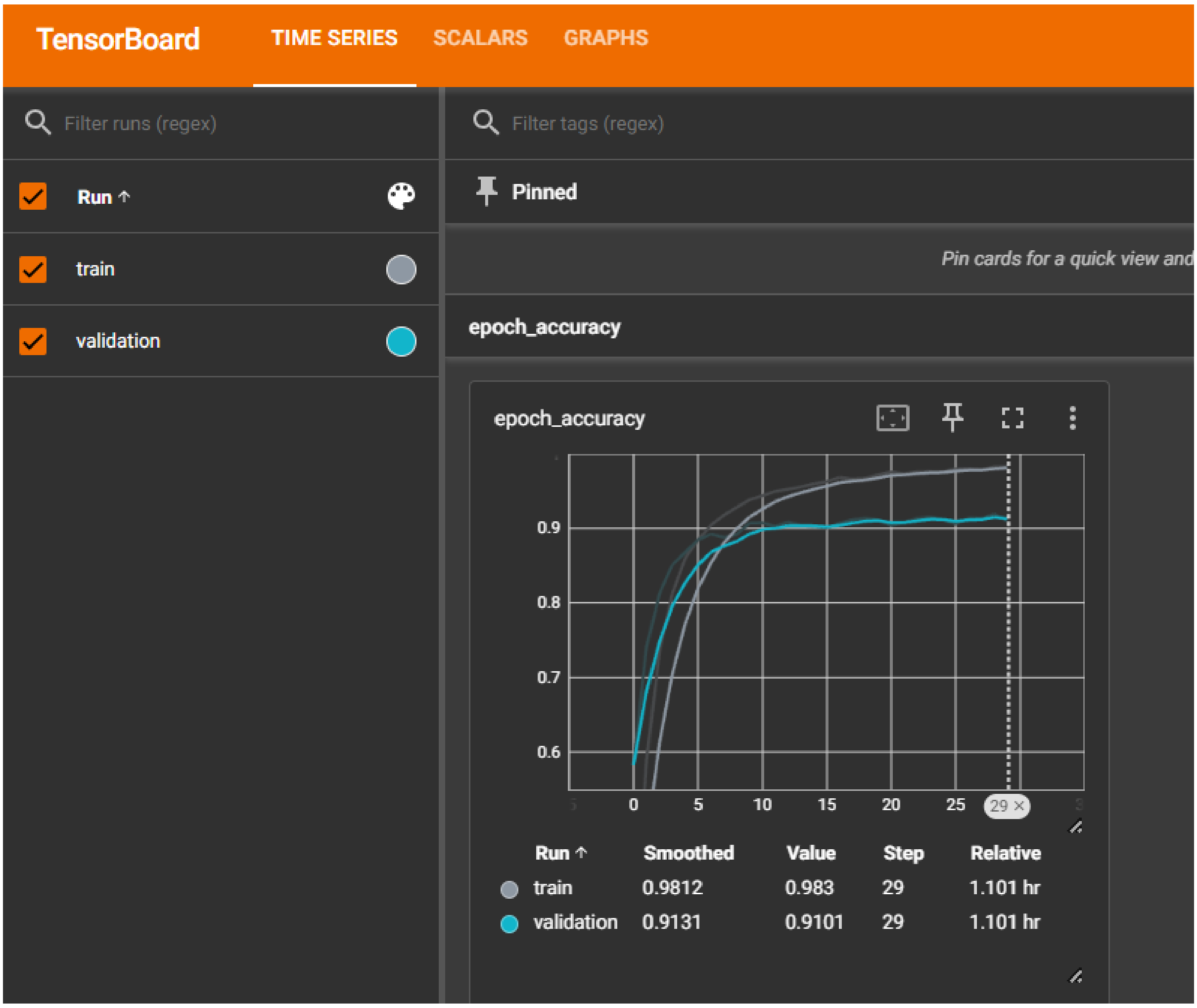
Weight decay comparison

Weight decay = 0.00001

```
print("------------Weight Decay Experiments-----------------")
weight_decays = [ 0.00001,0.0001, 0.001]  # Add your weight decay values
#fixed_optimizer = TrainingConfig.OPTIMIZERS[0]


weight0_model = training(
      **fixed_parameters,
      optimizer=keras.optimizers.Adam(learning_rate=TrainingConfig.LEARNING_RATE, weight_decay= weight_decays[0]),
      Learning_rate = TrainingConfig.LEARNING_RATE,
      train_ds=train_ds,
      val_ds=test_ds,
      model_name="weight_decay_" + str(weight_decays[0]),
      weight_decay=weight_decays[0],lr_schedule=None
)
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 32)	320
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
dropout (Dropout)	(None, 128, 128, 32)	0
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_1 (Dropout)	(None, 64, 64, 64)	0
conv2d_4 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_2 (Dropout)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 512)	33,554,944
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/30



Epoch 1/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **153s** 173ms/step - accuracy: 0.1971 - loss: 6.3485 - val_accuracy: 0.5824 - val_loss: 1.5951

Epoch 2/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **116s** 153ms/step - accuracy: 0.5312 - loss: 1.5484 - val_accuracy: 0.7405 - val_loss: 0.8692

Epoch 3/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **147s** 160ms/step - accuracy: 0.7100 - loss: 0.9109 - val_accuracy: 0.8132 - val_loss: 0.5883

Epoch 4/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **136s** 152ms/step - accuracy: 0.8049 - loss: 0.5994 - val_accuracy: 0.8516 - val_loss: 0.4855

Epoch 5/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 147ms/step - accuracy: 0.8537 - loss: 0.4490 - val_accuracy: 0.8675 - val_loss: 0.4138

Epoch 6/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **116s** 154ms/step - accuracy: 0.8761 - loss: 0.3685 - val_accuracy: 0.8845 - val_loss: 0.3599

Epoch 7/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **115s** 152ms/step - accuracy: 0.9028 - loss: 0.2918 - val_accuracy: 0.8925 - val_loss: 0.3491

Epoch 8/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **145s** 156ms/step - accuracy: 0.9182 - loss: 0.2436 - val_accuracy: 0.8877 - val_loss: 0.3459

Epoch 9/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **135s** 147ms/step - accuracy: 0.9257 - loss: 0.2257 - val_accuracy: 0.8916 - val_loss: 0.3420

Epoch 10/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **149s** 156ms/step - accuracy: 0.9357 - loss: 0.1966 - val_accuracy: 0.9070 - val_loss: 0.3095

Epoch 11/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **134s** 146ms/step - accuracy: 0.9410 - loss: 0.1769 - val_accuracy: 0.9076 - val_loss: 0.3123

Epoch 12/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9472 - loss: 0.1655 - val_accuracy: 0.9027 - val_loss: 0.3227

Epoch 13/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **149s** 156ms/step - accuracy: 0.9523 - loss: 0.1419 - val_accuracy: 0.9088 - val_loss: 0.3045

Epoch 14/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **132s** 143ms/step - accuracy: 0.9557 - loss: 0.1397 - val_accuracy: 0.9037 - val_loss: 0.3195

Epoch 15/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **144s** 146ms/step - accuracy: 0.9587 - loss: 0.1284 - val_accuracy: 0.9029 - val_loss: 0.3105

Epoch 16/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9601 - loss: 0.1211 - val_accuracy: 0.9004 - val_loss: 0.3359

Epoch 17/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9683 - loss: 0.0987 - val_accuracy: 0.9072 - val_loss: 0.3092

Epoch 18/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **106s** 140ms/step - accuracy: 0.9668 - loss: 0.1036 - val_accuracy: 0.9118 - val_loss: 0.3154

Epoch 19/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **150s** 151ms/step - accuracy: 0.9672 - loss: 0.1009 - val_accuracy: 0.9138 - val_loss: 0.3019

Epoch 20/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **138s** 146ms/step - accuracy: 0.9729 - loss: 0.0843 - val_accuracy: 0.9114 - val_loss: 0.3116

Epoch 21/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **140s** 143ms/step - accuracy: 0.9748 - loss: 0.0768 - val_accuracy: 0.9037 - val_loss: 0.4259

Epoch 22/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **141s** 142ms/step - accuracy: 0.9728 - loss: 0.0880 - val_accuracy: 0.9083 - val_loss: 0.3353

Epoch 23/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **141s** 140ms/step - accuracy: 0.9759 - loss: 0.0768 - val_accuracy: 0.9136 - val_loss: 0.3068

Epoch 24/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 140ms/step - accuracy: 0.9752 - loss: 0.0754 - val_accuracy: 0.9154 - val_loss: 0.3318

Epoch 25/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **144s** 143ms/step - accuracy: 0.9764 - loss: 0.0735 - val_accuracy: 0.9118 - val_loss: 0.3776

Epoch 26/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **110s** 145ms/step - accuracy: 0.9798 - loss: 0.0627 - val_accuracy: 0.9058 - val_loss: 0.3982

Epoch 27/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9773 - loss: 0.0708 - val_accuracy: 0.9143 - val_loss: 0.3637

Epoch 28/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **138s** 141ms/step - accuracy: 0.9778 - loss: 0.0735 - val_accuracy: 0.9129 - val_loss: 0.3501

Epoch 29/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **146s** 146ms/step - accuracy: 0.9828 - loss: 0.0559 - val_accuracy: 0.9195 - val_loss: 0.3791

Epoch 30/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9849 - loss: 0.0519 - val_accuracy: 0.9101 - val_loss: 0.3776

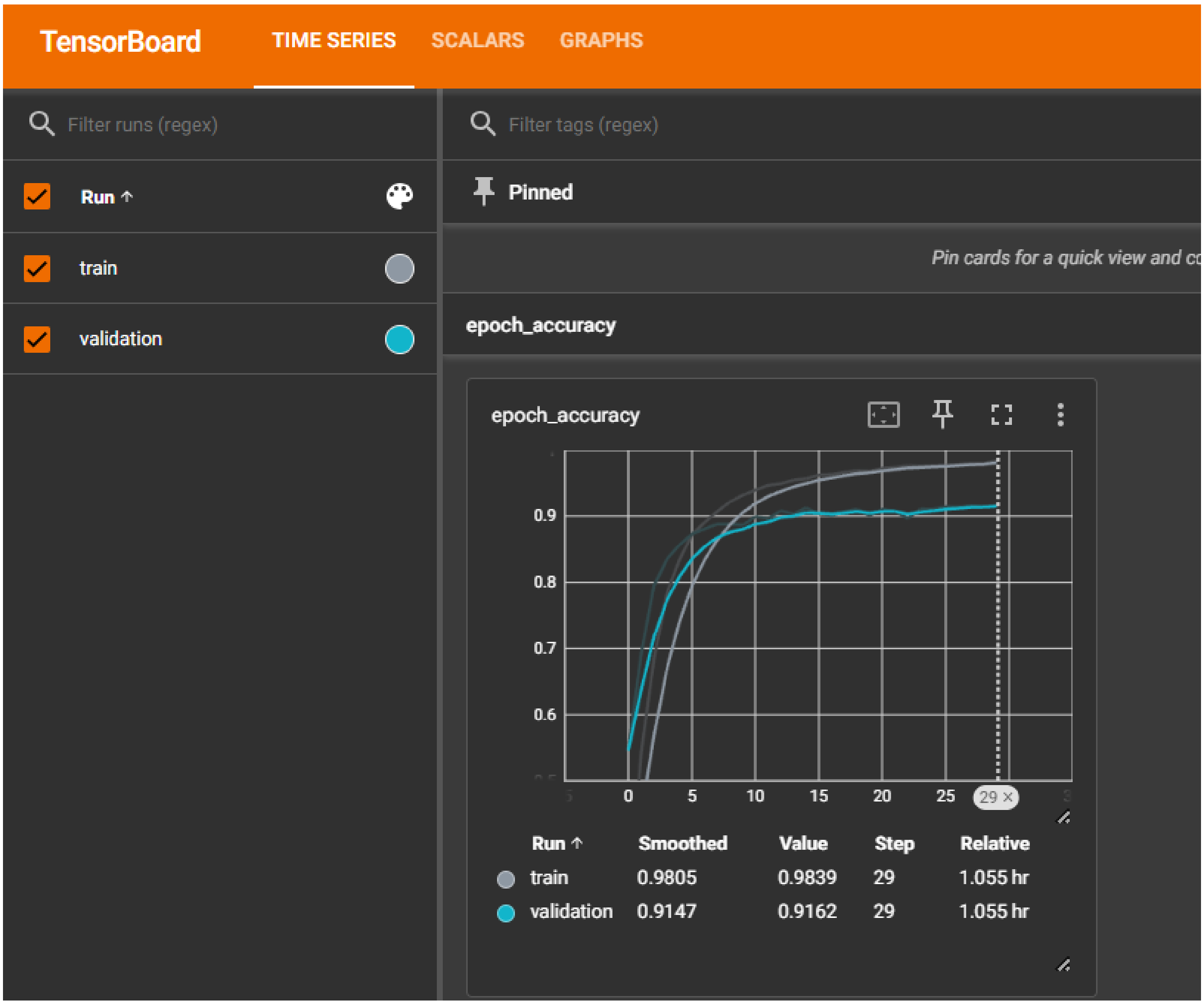
Weight decay = 0.0001

```
weight1_model = training(
        **fixed_parameters,
        optimizer=keras.optimizers.Adam(learning_rate=TrainingConfig.LEARNING_RATE, weight_decay =weight_decays[1]),
        Learning_rate = TrainingConfig.LEARNING_RATE,
        train_ds=train_ds,
        val_ds=test_ds,
        model_name="weight_decay_" + str(weight_decays[1]),
        weight_decay=weight_decays[1],lr_schedule=None
)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 256, 256, 32)	320
conv2d_7 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d_3 (MaxPooling2D)	(None, 128, 128, 32)	0
dropout_4 (Dropout)	(None, 128, 128, 32)	0
conv2d_8 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_9 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_5 (Dropout)	(None, 64, 64, 64)	0
conv2d_10 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_11 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_6 (Dropout)	(None, 32, 32, 64)	0
flatten_1 (Flatten)	(None, 65536)	0
dense_2 (Dense)	(None, 512)	33,554,944
dropout_7 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)



Epoch 1/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **122s** 154ms/step - accuracy: 0.1718 - loss: 6.3073 - val_accuracy: 0.5457 - val_loss: 1.5614

Epoch 2/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **116s** 152ms/step - accuracy: 0.4957 - loss: 1.6900 - val_accuracy: 0.6917 - val_loss: 1.0750

Epoch 3/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **145s** 156ms/step - accuracy: 0.6509 - loss: 1.1071 - val_accuracy: 0.7956 - val_loss: 0.6627

Epoch 4/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 156ms/step - accuracy: 0.7644 - loss: 0.7320 - val_accuracy: 0.8341 - val_loss: 0.5072

Epoch 5/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 156ms/step - accuracy: 0.8192 - loss: 0.5441 - val_accuracy: 0.8561 - val_loss: 0.4312

Epoch 6/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 152ms/step - accuracy: 0.8639 - loss: 0.4052 - val_accuracy: 0.8733 - val_loss: 0.3773

Epoch 7/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **118s** 156ms/step - accuracy: 0.8853 - loss: 0.3437 - val_accuracy: 0.8811 - val_loss: 0.3724

Epoch 8/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 152ms/step - accuracy: 0.9025 - loss: 0.2933 - val_accuracy: 0.8880 - val_loss: 0.3385

Epoch 9/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **138s** 146ms/step - accuracy: 0.9179 - loss: 0.2495 - val_accuracy: 0.8880 - val_loss: 0.3547

Epoch 10/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9293 - loss: 0.2134 - val_accuracy: 0.8868 - val_loss: 0.3530

Epoch 11/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **146s** 151ms/step - accuracy: 0.9389 - loss: 0.1882 - val_accuracy: 0.8994 - val_loss: 0.3260

Epoch 12/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **117s** 154ms/step - accuracy: 0.9451 - loss: 0.1699 - val_accuracy: 0.8964 - val_loss: 0.3187

Epoch 13/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **138s** 148ms/step - accuracy: 0.9506 - loss: 0.1493 - val_accuracy: 0.9080 - val_loss: 0.2911

Epoch 14/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **136s** 141ms/step - accuracy: 0.9556 - loss: 0.1374 - val_accuracy: 0.9032 - val_loss: 0.3230

Epoch 15/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **146s** 146ms/step - accuracy: 0.9545 - loss: 0.1344 - val_accuracy: 0.9126 - val_loss: 0.2946

Epoch 16/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **110s** 146ms/step - accuracy: 0.9632 - loss: 0.1137 - val_accuracy: 0.9034 - val_loss: 0.3350

Epoch 17/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 146ms/step - accuracy: 0.9627 - loss: 0.1167 - val_accuracy: 0.9012 - val_loss: 0.3567

Epoch 18/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **140s** 143ms/step - accuracy: 0.9625 - loss: 0.1163 - val_accuracy: 0.9067 - val_loss: 0.3093

Epoch 19/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **110s** 145ms/step - accuracy: 0.9687 - loss: 0.0929 - val_accuracy: 0.9103 - val_loss: 0.3089

Epoch 20/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **107s** 141ms/step - accuracy: 0.9691 - loss: 0.1005 - val_accuracy: 0.9010 - val_loss: 0.3634

Epoch 21/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **107s** 141ms/step - accuracy: 0.9719 - loss: 0.0877 - val_accuracy: 0.9105 - val_loss: 0.3185

Epoch 22/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **145s** 145ms/step - accuracy: 0.9722 - loss: 0.0870 - val_accuracy: 0.9075 - val_loss: 0.3536

Epoch 23/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **110s** 145ms/step - accuracy: 0.9777 - loss: 0.0680 - val_accuracy: 0.8964 - val_loss: 0.5002

Epoch 24/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **110s** 145ms/step - accuracy: 0.9714 - loss: 0.0945 - val_accuracy: 0.9106 - val_loss: 0.3268

Epoch 25/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 145ms/step - accuracy: 0.9759 - loss: 0.0726 - val_accuracy: 0.9103 - val_loss: 0.3522

Epoch 26/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 145ms/step - accuracy: 0.9780 - loss: 0.0705 - val_accuracy: 0.9143 - val_loss: 0.3275

Epoch 27/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **138s** 140ms/step - accuracy: 0.9758 - loss: 0.0708 - val_accuracy: 0.9136 - val_loss: 0.3128

Epoch 28/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 140ms/step - accuracy: 0.9811 - loss: 0.0592 - val_accuracy: 0.9156 - val_loss: 0.3593

Epoch 29/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **107s** 141ms/step - accuracy: 0.9802 - loss: 0.0657 - val_accuracy: 0.9143 - val_loss: 0.3448

Epoch 30/30

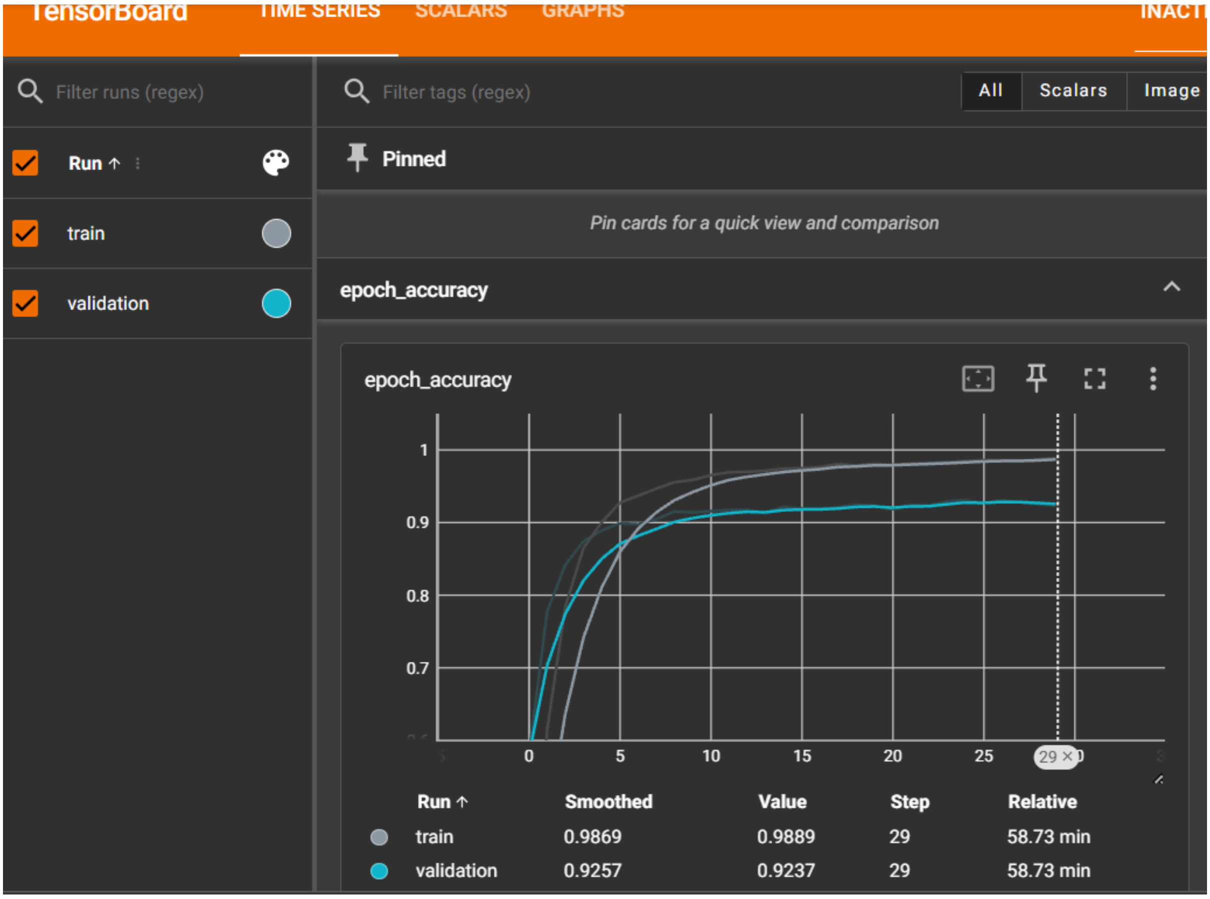
757/757 ━━━━━━━━━━━━━━━━━━━━ **143s** 143ms/step - accuracy: 0.9832 - loss: 0.0522 - val_accuracy: 0.9162 - val_loss: 0.3556

	Accuaracy	Precision	Recall	F1_Score
Decay_0_00001_train	0.999339	0.999343	0.999339	0.999339
Decay_0_00001_valid	0.913762	0.916989	0.913762	0.914014
Decay_0_0001_train	0.997481	0.997505	0.997481	0.997482
Decay_0_0001_valid	0.907980	0.910788	0.907980	0.908190

Batch comparison

Batch size = 32

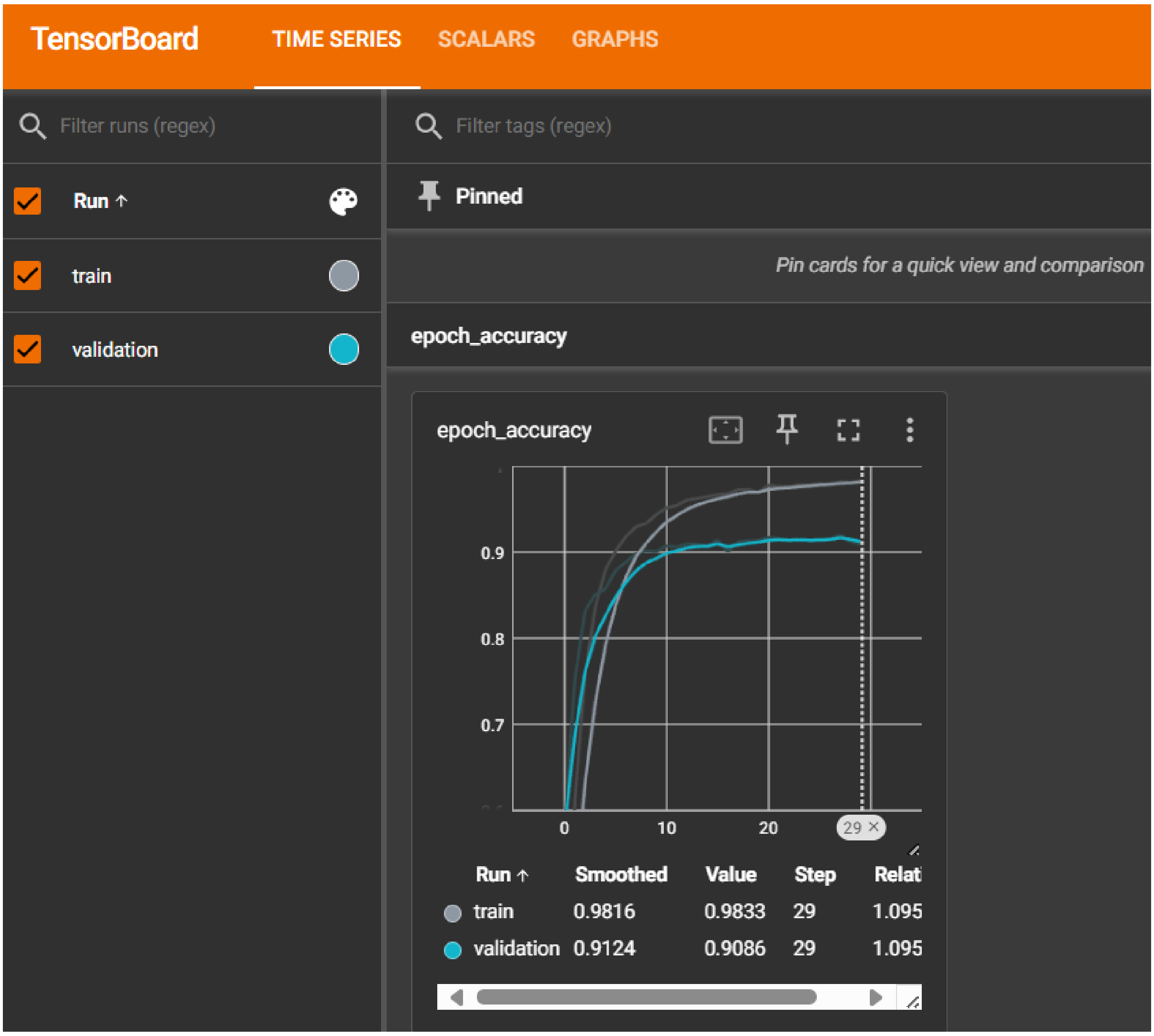
Batch size = 128



	Accuaracy	Precision	Recall	F1_Score
batch_128_train	0.999959	0.999959	0.999959	0.999959
batch_128_valid	0.930448	0.932061	0.930448	0.930101
batch_32_train	0.124463	0.178456	0.124463	0.083265
batch_32_valid	0.115315	0.139571	0.115315	0.076174

Number of convolution layers comparison

Number of layers = 2



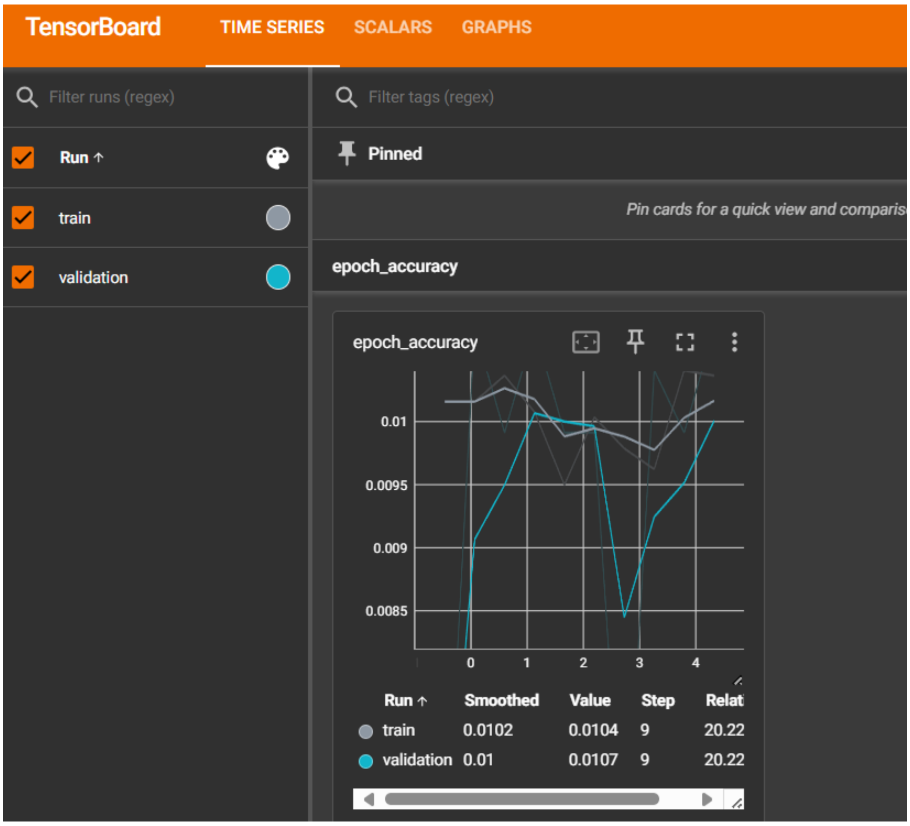
Learning rate comparison

learning rate = 0.1

```
Adam_model_0_point_1 = training(2, 3, 0.25, keras.optimizers.Adam(learning_rate=0.1), TrainingConfig.LEARNING_RATE, train_ds, test_ds,"ADAM_optimizer_0.1")
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 32)	320
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
dropout (Dropout)	(None, 128, 128, 32)	0
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_1 (Dropout)	(None, 64, 64, 64)	0
conv2d_4 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_2 (Dropout)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 512)	33,554,944
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)



Epoch 1/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **156s** 177ms/step - accuracy: 0.0102 - loss: 10515372.0000 - val_accuracy: 0.0063 - val_loss: 4.6442

Epoch 2/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **112s** 148ms/step - accuracy: 0.0108 - loss: 4.6396 - val_accuracy: 0.0107 - val_loss: 4.6524

Epoch 3/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 144ms/step - accuracy: 0.0118 - loss: 4.6388 - val_accuracy: 0.0099 - val_loss: 4.6453

Epoch 4/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **148s** 152ms/step - accuracy: 0.0104 - loss: 4.6392 - val_accuracy: 0.0107 - val_loss: 4.6425

Epoch 5/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 149ms/step - accuracy: 0.0104 - loss: 4.6380 - val_accuracy: 0.0099 - val_loss: 4.6390

Epoch 6/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **109s** 144ms/step - accuracy: 0.0104 - loss: 4.6407 - val_accuracy: 0.0099 - val_loss: 4.6466

Epoch 7/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 140ms/step - accuracy: 0.0108 - loss: 4.6389 - val_accuracy: 0.0063 - val_loss: 4.6474

Epoch 8/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **141s** 139ms/step - accuracy: 0.0100 - loss: 4.6400 - val_accuracy: 0.0104 - val_loss: 4.6470

Epoch 9/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 139ms/step - accuracy: 0.0110 - loss: 4.6380 - val_accuracy: 0.0099 - val_loss: 4.6424

Epoch 10/30

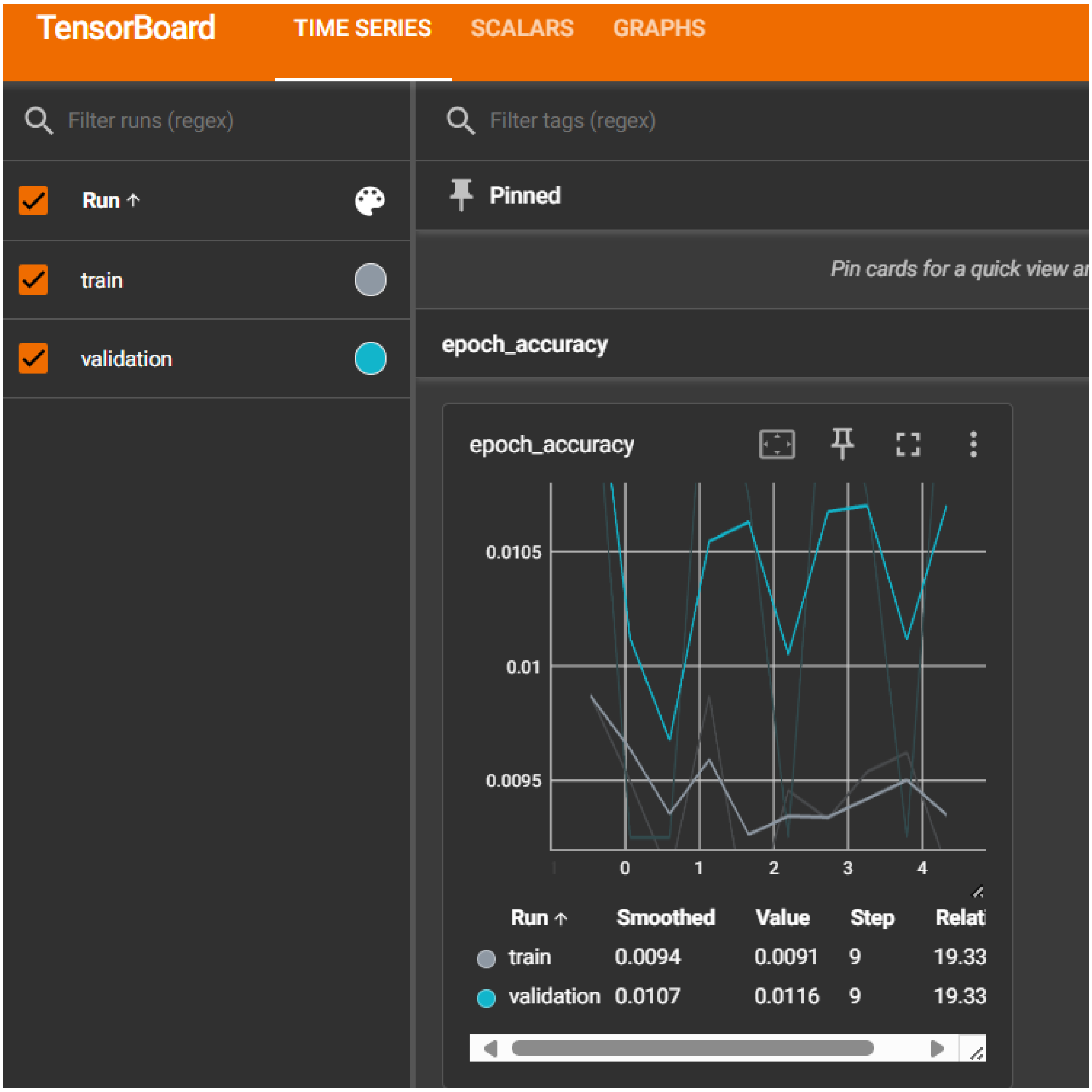
757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 140ms/step - accuracy: 0.0114 - loss: 4.6376 - val_accuracy: 0.0107 - val_loss: 4.6486

learning rate = 0.01

```
Adam_model_0_point_01 = training(2, 3, 0.25, keras.optimizers.Adam(learning_rate=0.01), TrainingConfig.LEARNING_RATE, train_ds, test_ds,"ADAM_optimizer_0.01")
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 256, 256, 32)	320
conv2d_7 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d_3 (MaxPooling2D)	(None, 128, 128, 32)	0
dropout_4 (Dropout)	(None, 128, 128, 32)	0
conv2d_8 (Conv2D)	(None, 128, 128, 64)	18,496
conv2d_9 (Conv2D)	(None, 128, 128, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_5 (Dropout)	(None, 64, 64, 64)	0
conv2d_10 (Conv2D)	(None, 64, 64, 64)	36,928
conv2d_11 (Conv2D)	(None, 64, 64, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_6 (Dropout)	(None, 32, 32, 64)	0
flatten_1 (Flatten)	(None, 65536)	0
dense_2 (Dense)	(None, 512)	33,554,944
dropout_7 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 99)	50,787

Total params: 33,744,579 (128.73 MB)
Trainable params: 33,744,579 (128.73 MB)
Non-trainable params: 0 (0.00 B)



Epoch 1/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **124s** 155ms/step - accuracy: 0.0109 - loss: 23.4048 - val_accuracy: 0.0116 - val_loss: 4.6016

Epoch 2/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **130s** 144ms/step - accuracy: 0.0096 - loss: 4.6003 - val_accuracy: 0.0093 - val_loss: 4.6018

Epoch 3/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **104s** 138ms/step - accuracy: 0.0098 - loss: 4.6004 - val_accuracy: 0.0093 - val_loss: 4.6018

Epoch 4/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **110s** 145ms/step - accuracy: 0.0102 - loss: 4.6005 - val_accuracy: 0.0116 - val_loss: 4.6016

Epoch 5/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **141s** 144ms/step - accuracy: 0.0090 - loss: 4.6005 - val_accuracy: 0.0107 - val_loss: 4.6018

Epoch 6/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **139s** 141ms/step - accuracy: 0.0095 - loss: 4.6005 - val_accuracy: 0.0093 - val_loss: 4.6018

Epoch 7/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **108s** 143ms/step - accuracy: 0.0096 - loss: 4.6007 - val_accuracy: 0.0116 - val_loss: 4.6019

Epoch 8/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **140s** 140ms/step - accuracy: 0.0103 - loss: 4.6006 - val_accuracy: 0.0107 - val_loss: 4.6022

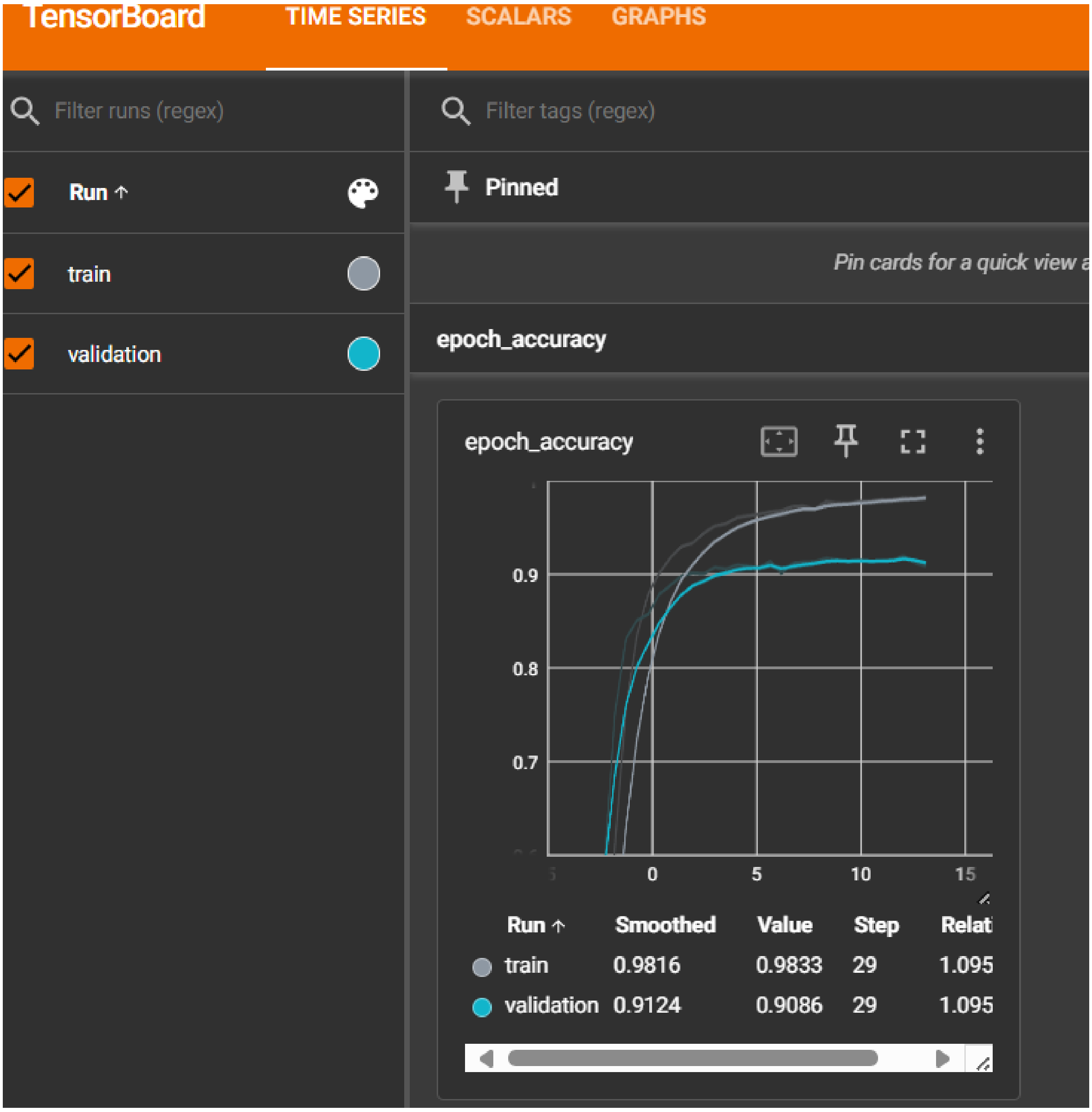
Epoch 9/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **140s** 138ms/step - accuracy: 0.0102 - loss: 4.6004 - val_accuracy: 0.0093 - val_loss: 4.6020

Epoch 10/30

757/757 ━━━━━━━━━━━━━━━━━━━━ **142s** 138ms/step - accuracy: 0.0093 - loss: 4.6006 - val_accuracy: 0.0116 - val_loss: 4.6017

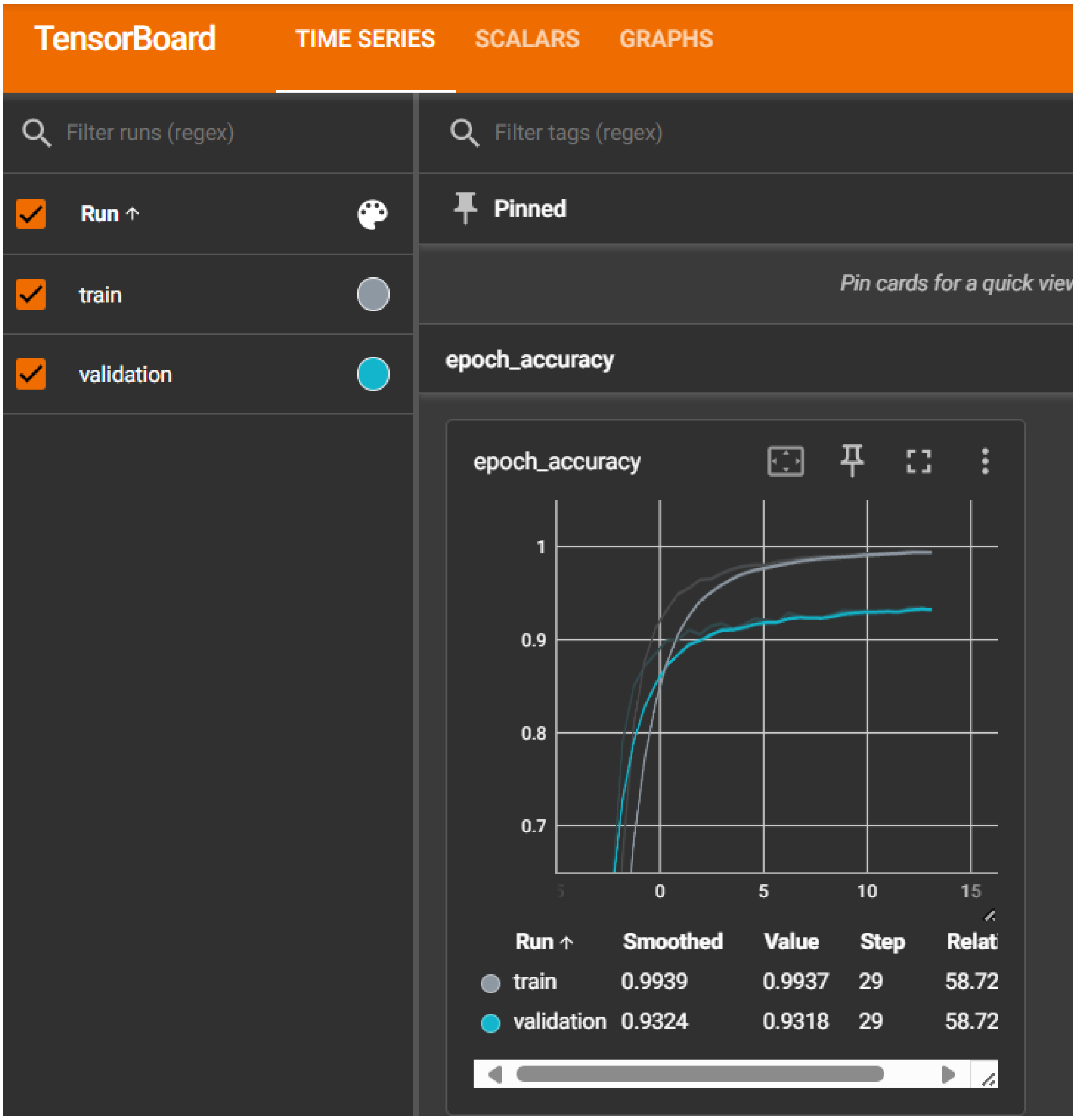
learning rate = 0.001



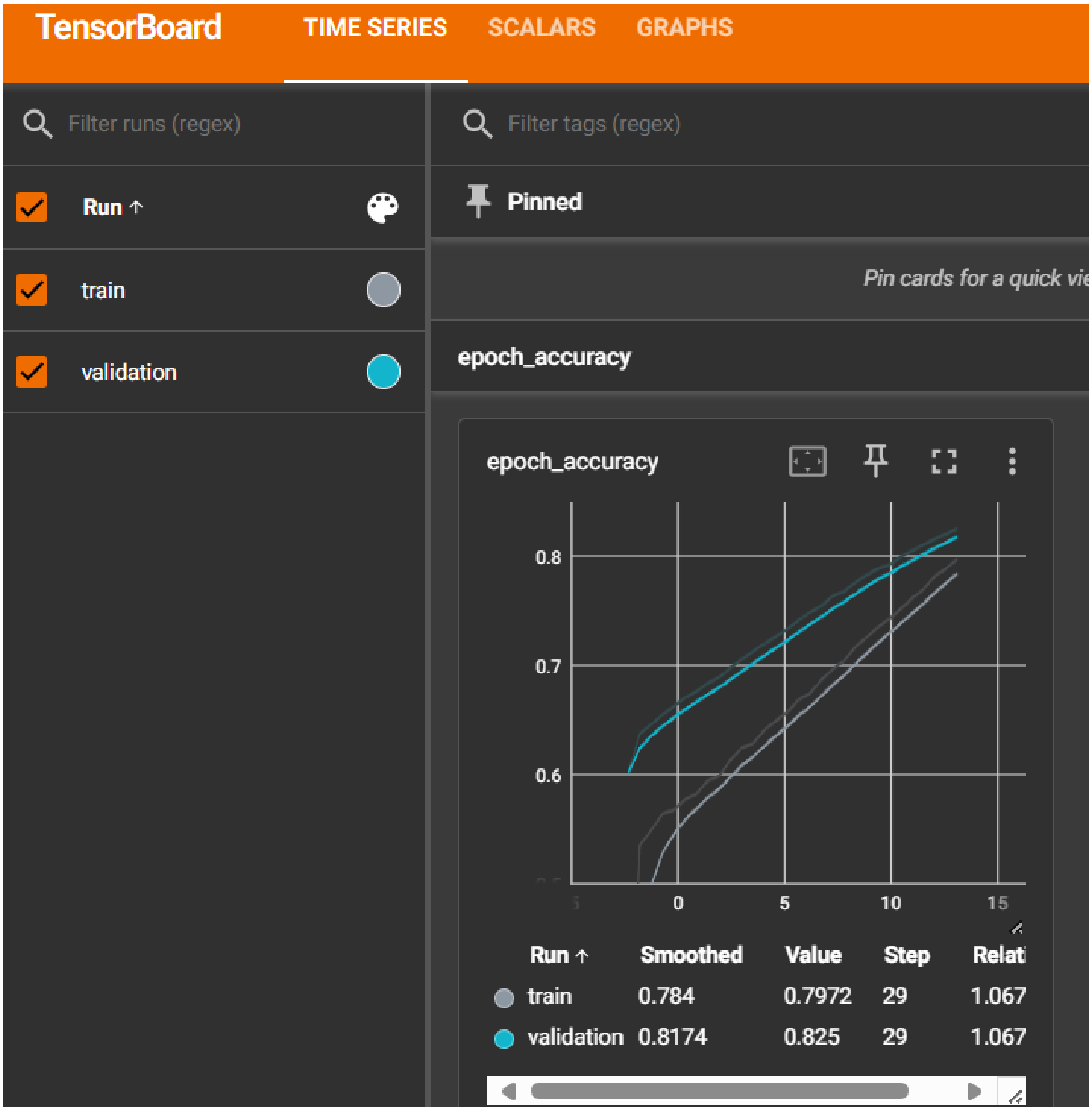
	Accuaracy	Precision	Recall	F1_Score
Adam_0_point_1_train	0.010282	0.000106	0.010282	0.000209
Adam_0_point_1_valid	0.009912	0.000098	0.009912	0.000195
Adam_0_point_01_train	0.009663	0.000093	0.009663	0.000185
Adam_0_point_01_valid	0.011565	0.000134	0.011565	0.000264
Adam_0_point_001_train	0.009663	0.000093	0.009663	0.000185
Adam_0_point_001_valid	0.011565	0.000134	0.011565	0.000264

Schedulers Comparison

Exponential Decay



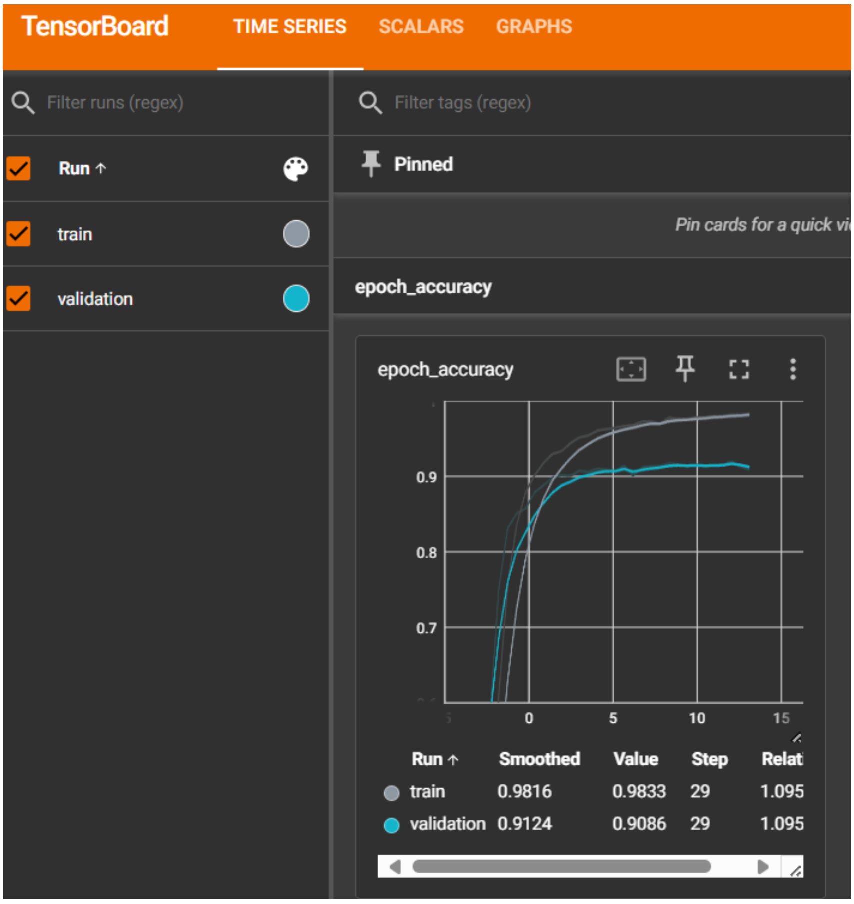
Cosin Decay



	Accuaracy	Precision	Recall	F1_Score
Exponetial_schedualr_train	0.862942	0.902077	0.862942	0.853987
Exponetial_schedualr_valid	0.715678	0.782857	0.715678	0.698439
Cosine_schedualr_train	0.944582	0.945653	0.944582	0.944273
Cosine_schedualr_valid	0.826037	0.831238	0.826037	0.824975

Dropout Comparison

Dropout = 0.25



	Accuracy	Precision	Recall	F1_Score
Dropout_0_1_train	0.999215	0.999221	0.999215	0.999215
Dropout_0_1_valid	0.929787	0.933082	0.929787	0.929831
Dropout_0_25_train	0.124463	0.178456	0.124463	0.083265
Dropout_0_25_valid	0.115315	0.139571	0.115315	0.076174
Dropout_0_4_train	0.998720	0.998722	0.998720	0.998719
Dropout_0_4_valid	0.933421	0.935520	0.933421	0.933372

CSV classification

```
trainCSV = pd.read_csv(r"/content/train.csv")
testCSV = pd.read_csv(r"/content/test.csv")
```

Loading the data from the CSV file

```
text_features = trainCSV.select_dtypes(include=['object']).columns.tolist()
labelEncoder = LabelEncoder()
for i in text_features:
    labelEncoder.fit(trainCSV[i])
    trainCSV[i] = labelEncoder.transform(trainCSV[i])

outputTrain = pd.DataFrame(trainCSV['species'])
trainCSV.drop(columns=['id'],inplace=True)
trainCSV.drop(columns=['species'],inplace=True)

normalized_data = MinMaxScaler(feature_range=(-1,1))
normalized_data.fit(trainCSV)
dfTrain = normalized_data.transform(trainCSV)
```

Taking the categorical data and changing it to numerical then removing the id column and taking the species column and storing it in the output then also removing it then normalizing all the numeric features so they can be within the range of -1 and 1

```
from sklearn.decomposition import PCA
pca = PCA(n_components=98)
pca.fit(dfTrain)
X_pca = pca.transform(dfTrain)
```

Using PCA for feature reduction to reduce feature from 192 to 98

```
lda = LinearDiscriminantAnalysis(n_components=98)
X_train = lda.fit_transform(dfTrain, outputTrain)
X_test = lda.transform(dfTrain)
print(X_train.shape)
```

Using LDA for feature reduction to reduce feature from 192 to 98

```
outputTrain = outputTrain.to_numpy(dtype=np.float32)
```

```
outputTrain = to_categorical(outputTrain, num_classes=99)
```

```
X_train1,X_test1,y_train1,y_test1 = train_test_split(X_train,outputTrain,test_size=0.2)
```

```
X_train2,X_test2,y_train2,y_test2 = train_test_split(X_pca,outputTrain,test_size=0.2)
```

Changing the shape of the output so in can be read by the CNN model then splitting the data into train and test

```
model = Sequential([
    Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(98, 1)),
    Conv1D(filters=32, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.25),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(99, activation='softmax')
])
```

Creating a CNN model with 1D convolutions and 1D Maxpooling as the CSV data is 1D the model consists of two 32 convolution filters then maxpooling 2 then a dropout of 25% then two 64 convolution filters then maxpooling 2 then a dropout of 25% then flatten and dense to give output as one of the class which are 99 class

Using LDA data

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

model.fit(X_train1, y_train1, epochs=30, batch_size=32)
```

Epoch 1/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **3s** 15ms/step - accuracy: 0.0180 - loss: 4.6022

Epoch 2/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - accuracy: 0.0567 - loss: 4.4626

Epoch 3/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.2731 - loss: 3.5025

Epoch 4/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.6863 - loss: 1.2285

Epoch 5/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.8343 - loss: 0.6227

Epoch 6/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9644 - loss: 0.1820

Epoch 7/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 17ms/step - accuracy: 0.9512 - loss: 0.1565

Epoch 8/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - accuracy: 0.9717 - loss: 0.1096

Epoch 9/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 18ms/step - accuracy: 0.9682 - loss: 0.1257

Epoch 10/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 18ms/step - accuracy: 0.9946 - loss: 0.0311

Epoch 11/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 26ms/step - accuracy: 0.9899 - loss: 0.0300

Epoch 12/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 26ms/step - accuracy: 0.9779 - loss: 0.1005

Epoch 13/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 28ms/step - accuracy: 0.9930 - loss: 0.0294

Epoch 14/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9929 - loss: 0.0214

Epoch 15/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9863 - loss: 0.0350

Epoch 16/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - accuracy: 0.9806 - loss: 0.0634

Epoch 17/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9928 - loss: 0.0315

Epoch 18/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - accuracy: 0.9996 - loss: 0.0113

Epoch 19/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9963 - loss: 0.0113

Epoch 20/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9945 - loss: 0.0161

Epoch 21/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9986 - loss: 0.0109

Epoch 22/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9949 - loss: 0.0145

Epoch 23/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 1.0000 - loss: 0.0038

Epoch 24/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9970 - loss: 0.0076

Epoch 25/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9981 - loss: 0.0155

Epoch 26/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9906 - loss: 0.0524

Epoch 27/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - accuracy: 0.9977 - loss: 0.0108

Epoch 28/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - accuracy: 1.0000 - loss: 0.0046

Epoch 29/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 1.0000 - loss: 0.0021

Epoch 30/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 1.0000 - loss: 0.0025

```
loss, accuracy = model.evaluate(X_test1, y_test1)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9710 - loss: 0.0803
Test Accuracy: 95.96%
```

Using the optimizer as adam and the LDA features training accuracy was 100% and the final validation accuracy was 95.96%

Using PCA data

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

model.fit(X_train2, y_train2, epochs=30, batch_size=32)
```

Epoch 1/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **3s** 17ms/step - accuracy: 0.0334 - loss: 4.6960

Epoch 2/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.1245 - loss: 3.9559

Epoch 3/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - accuracy: 0.3740 - loss: 2.5710

Epoch 4/30

25/25 **0s** 16ms/step - accuracy: 0.7208 - loss: 1.2443

Epoch 5/30

25/25 **1s** 15ms/step - accuracy: 0.8674 - loss: 0.5069

Epoch 6/30

25/25 **1s** 15ms/step - accuracy: 0.9200 - loss: 0.2893

Epoch 7/30

25/25 **0s** 16ms/step - accuracy: 0.9716 - loss: 0.1168

Epoch 8/30

25/25 **1s** 15ms/step - accuracy: 0.9868 - loss: 0.0540

Epoch 9/30

25/25 **0s** 16ms/step - accuracy: 0.9909 - loss: 0.0454

Epoch 10/30

25/25 **1s** 15ms/step - accuracy: 0.9844 - loss: 0.0605

Epoch 11/30

25/25 **1s** 16ms/step - accuracy: 0.9884 - loss: 0.0510

Epoch 12/30

25/25 **0s** 15ms/step - accuracy: 0.9748 - loss: 0.0951

Epoch 13/30

25/25 **1s** 16ms/step - accuracy: 0.9868 - loss: 0.0425

Epoch 14/30

25/25 **1s** 14ms/step - accuracy: 0.9961 - loss: 0.0264

Epoch 15/30

25/25 **1s** 15ms/step - accuracy: 0.9897 - loss: 0.0345

Epoch 16/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9957 - loss: 0.0211

Epoch 17/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 14ms/step - accuracy: 0.9944 - loss: 0.0210

Epoch 18/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - accuracy: 0.9861 - loss: 0.0301

Epoch 19/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 23ms/step - accuracy: 0.9983 - loss: 0.0192

Epoch 20/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 27ms/step - accuracy: 0.9987 - loss: 0.0085

Epoch 21/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 27ms/step - accuracy: 0.9982 - loss: 0.0142

Epoch 22/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 27ms/step - accuracy: 0.9984 - loss: 0.0096

Epoch 23/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 1.0000 - loss: 0.0079

Epoch 24/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - accuracy: 1.0000 - loss: 0.0063

Epoch 25/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9923 - loss: 0.0149

Epoch 26/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9871 - loss: 0.0308

Epoch 27/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9917 - loss: 0.0295

Epoch 28/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - accuracy: 0.9928 - loss: 0.0250

Epoch 29/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9903 - loss: 0.0273

Epoch 30/30

25/25 ━━━━━━━━━━━━━━━━━━━━ **1s** 15ms/step - accuracy: 0.9974 - loss: 0.0131

```
loss, accuracy = model.evaluate(X_test2, y_test2)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8936 - loss: 0.5956
Test Accuracy: 88.38%
```

Using the optimizer as adam and the PCA features training accuracy was 99.74% and the final validation accuracy was 88.38%

NoteBook Link:

Mohamed_Leaf_Classification.ipynb - Colab