پروژه اول درس آنالیز داده

آنوشا شریعتی 9923041

املین غازاریان 9923056

بخش 1: آشنایی با داده و پیش پردازش:

قسمت 1:

در ابتدا کتابخانه های مورد نیاز را اضافه میکنیم سپس فایل دیتاست را بارگذاری کرده و از حالت زیپ خارج ساخته و آن را به صورت زیر آماده سازی میکنیم.

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import torch.nn as nn
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import os



print("GPU available: {}".format(torch.cuda.is_available()))
from google.colab import files
uploaded = files.upload()

# Extract the contents of the .zip file
import zipfile
import io
with zipfile.ZipFile("/content/brain_tumor_dataset.zip", 'r') as zip_ref:
    zip_ref.extractall('/content/dataset')
    zip_ref.close()

gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
data_dir = '/content/dataset/brain_tumor_dataset'
```

سپس با استفاده از دستورات زیر تصویر ها با پسوندهای قابل قبول را خوانده و با استفاده از کتابخانه کراس سایز آن ها را یکی کرده و درون متغیر دیتا لود میکنیم.

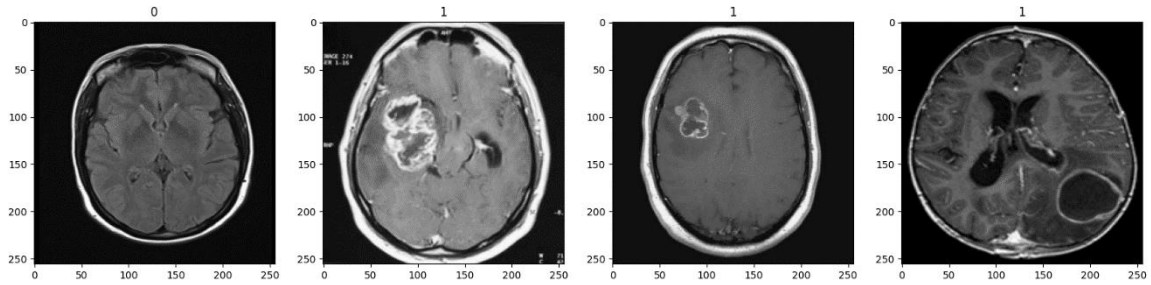
```
image_exts = ['jpeg','jpg', 'bmp', 'png']

for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))

data = tf.keras.utils.image_dataset_from_directory(data_dir)
```

با استفاده از دستورات زیر یک بچ 4 تایی از تصویر ها را لود کرده و نشان میدهیم. همان طور که مشاهده میشود کلاس0 مربوط به تصاویر بدون تومور وکلاس 1 مربوط به تومور مغزی است. و مشاهده میشود که سایز تصاویر هم یکی شده است.

```
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(batch[1][idx])
```



قسمت 2:

با استفاده از دستورهای زیر دیتا را با تقسیم بر 255 نورمالایز کرده و دیتا ست را به نصبت داده شده بین متغیر های آموزش، ولیدیشن و تست تقسیم میکنیم.

```
data = data.map(lambda x,y: (x/255, y))
scaled_iterator=data.as_numpy_iterator().next()
train_size = int(len(data)*.7)
val_size = int(len(data)*.15)
test_size = int(len(data)*.15)
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

بخش 2: آموزش مدل CNN

قسمت 1:

ابتدا کتاب خانه های مورد نیاز برای طراحی این مدل را اضافه کرده سپس مدل را به صورت زیر تعریف میکنیم.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,
Dropout, SeparableConv2D, Add, Input
from tensorflow.keras import layers

model = Sequential()
model.add(SeparableConv2D(32,   kernel_size=(3,   3),   activation='relu',
input_shape=(256,256,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(SeparableConv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

مدل تعریف شده به این صورت است:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 separable_conv2d_11 (Separ  (None, 254, 254, 32)      155
 ableConv2D)

 max_pooling2d_10 (MaxPooli  (None, 127, 127, 32)      0
 ng2D)

 dropout_13 (Dropout)        (None, 127, 127, 32)      0

 separable_conv2d_12 (Separ  (None, 125, 125, 64)      2400
 ableConv2D)

 max_pooling2d_11 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 dropout_14 (Dropout)        (None, 62, 62, 64)        0

 flatten_3 (Flatten)         (None, 246016)            0

 dense_10 (Dense)            (None, 256)               62980352

 dense_11 (Dense)            (None, 1)                 257

=================================================================
Total params: 62983164 (240.26 MB)
Trainable params: 62983164 (240.26 MB)
Non-trainable params: 0 (0.00 Byte)
```

- بلوک residual

بلوک‌های (Residual blocks) ابتکاری است که در شبکه‌های عصبی عمیق، به ویژه شبکه‌های CNN، مورد استفاده قرار می‌گیرد. این بلوک‌ها به طور خاص برای مقابله با مشکل ناپایداری و کاهش عملکرد شبکه‌ها در معماری‌های بسیار عمیق ابداع شده‌اند.

یک بلوک باقی‌مانده معمولاً شامل چندین لایه عصبی است که با یکدیگر ارتباط دارند. اما نکته مهم این است که بلوک باقی‌مانده، در میانه مسیر اصلی یک شبکه‌ی عصبی قرار دارد و یک ارتباط مستقیم (یا ارتباط هوشمندانه) از ابتدا تا انتهای بلوک، مستقیماً به خروجی بلوک اضافه می‌شود.

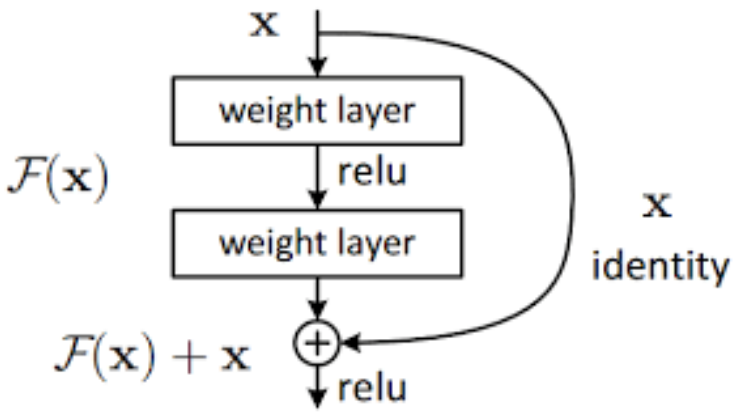
استفاده از بلوک‌های باقی‌مانده در شبکه‌های CNN برای کلاسیفیکیشن تصویر، به دلیل مزایای زیر است:

1. استفاده از گذرگاه‌های مستقیم (Shortcut Connections): اضافه کردن گذرگاه مستقیم از ورودی به خروجی بلوک، باعث می‌شود که اطلاعات اصلی و نادیده‌گرفتنی به صورت مستقیم منتقل شوند. این کمک می‌کند تا از ناپایداری یا کاهش عملکرد شبکه در طی آموزش جلوگیری شود.

2. مقابله با مشکل ناپایداری آموزش: در شبکه‌های عمیق، امکان دارد که آموزش به صورتی که بلوک‌های باقی‌مانده دچار ناپایداری شوند، ممکن است کارآیی شبکه را کاهش دهد. استفاده از بلوک‌های باقی‌مانده به شبکه‌ها کمک می‌کند تا به طور موثرتر و پایدارتر آموزش ببینند.

3. تسریع آموزش: اضافه کردن گذرگاه‌های مستقیم از ورودی به خروجی بلوک، می‌تواند منجر به شتاب دادن فرآیند آموزش شود. این کاهش می‌تواند باعث شود که شبکه‌ها به سرعت به دقت مطلوب برسند.

بنابراین، بلوک‌های باقی‌مانده در شبکه‌های CNN ابزاری موثر برای مقابله با مشکلاتی مانند بیش‌برازش، ناپایداری آموزش و تسریع فرآیند آموزش می‌باشند و در افزایش عملکرد و کارایی شبکه‌ها تأثیرگذارند.



برای اضافه کردن این بلوک تابعی را به صورت زیر تعریف کرده و در مدل اضافه کنیم.

```
from tensorflow.keras import layers

def residual_block(model, filters, kernel_size):
    y = SeparableConv2D(filters, kernel_size, activation='relu',
padding='same')(model.output)
    y = SeparableConv2D(filters, kernel_size, activation='relu',
padding='same')(y)
    y = Add()([model.output, y])  # Add skip connection
    y = MaxPooling2D(pool_size=(2, 2))(y)
    y = Dropout(0.25)(y)
    return y
```

```
model = Sequential()
model.add(Input(shape=(256, 256, 3)))

model.add(SeparableConv2D(32, kernel_size=(3,3), activation='relu',
padding='same'))

model.add(residual_block(model, filters=32, kernel_size=(3, 3)))

#model.add(residual_block(model.layers[-1].output, filters=128,
kernel_size=(3, 3)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

قسمت 2:

- تشخیص بیش برازش

برای تشخیص اینکه یک مدل CNN دچار بیش‌برازش شده است یا خیر، می‌تواند با بررسی رفتار نمودار لاس و دقت برای داده‌های آموزش و اعتبارسنجی انجام شود.

اگر لاس داده های آموزشی به طور پیوسته کاهش یابد اما لاس داده اعتبارسنجی پس از چند دوره شروع به افزایش یابد، نشان می‌دهد که مدل دچار بیش‌برازش شده است. این به این دلیل است که مدل داده آموزش را بسیار خوب حفظ می‌کند و ناتوان در تعمیم آن به داده‌هایی است که دیده نشده‌اند. اگر هم نمودار لاس داده آموزش و اعتبارسنجی به طور همزمان کاهش یا ثابت شوند، نشان می‌دهد که مدل به خوبی به داده‌های اعتبارسنجی تعمیم می‌یابد.

اگر دقت داده آموزشی به طور پیوسته افزایش یابد در حالی که دقت اعتبارسنجی شروع به کاهش یا ثابت شود، نشانه بیش‌برازش است. مدل دارد در شناسایی الگوها در داده‌های آموزش بسیار تخصصی می‌شود. اگر هم دقت آموزش و اعتبارسنجی به طور همزمان افزایش یا ثابت شوند، نشان می‌دهد که مدل به طور موثر بدون بیش‌برازش یاد می‌گیرد.

بیش‌برازش: اگر نشانه‌هایی از بیش‌برازش مشاهده شود (مانند افزایش لاس داده اعتبارسنجی یا کاهش دقت اعتبارسنجی در حالی که معیارهای آموزش بهبود می‌یابند)، می‌توان از تکنیک‌هایی مانند کاهش‌دهی، اعمال محدودیت وزن، یا افزایش داده‌های آموزش برای کاهش بیش‌برازش استفاده کرد

کم‌برازش: اگر هم لاس داده‌های آموزش و اعتبارسنجی همچنان بالا باقی بمانند و دقت‌ها همچنان پایین باشند، ممکن است کم‌برازش باشد. می‌توان این موضوع را با افزایش پیچیدگی مدل، تنظیم نرخ یادگیری، یا آموزش بیشتر مدل مرتفع کرد.

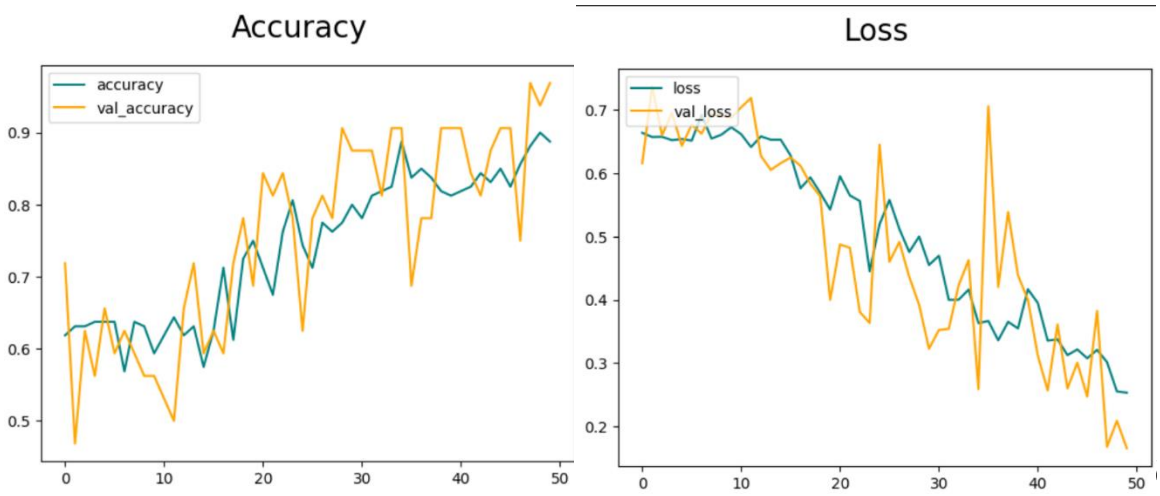
در ادامه برای بررسی موارد بالا مدل را اجرا کرده و مقادیر و منحنی های خواسته شده را رسم میکنیم.

```
model.compile('adam',loss=tf.losses.BinaryCrossentropy(),
metrics=['accuracy'])
logdir='logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
hist = model.fit(train, epochs=50, validation_data=val)
```

مقادیر loss , accuracy را برای 50 ایپاک محاسبه کردیم. مقدار دقت در نهایت به عدد 88 درصد رسید و نمودار تغییرات آن نشان میدهد که دقت در ابتدا در حدود 60 درصد بوده و سپس به 88 درصد رسیده است. نمودار دقت در داده های آموزشی و ولیدیشن یک روند ثابت و مشابه را طی میکند که نشانمیدهد مدل دچار overfit یا underfit نشده است.

```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()

fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



در این قسمت با اضافه کردن کد زیر معیار های ارزیابی خواسته شده بررسی شد.

```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report

# Predict labels for the test set
test_predictions = model.predict(test)
test_predictions = np.argmax(test_predictions, axis=-1)

# Extract true labels from the test set
test_labels = []
for _, label in test:
    test_labels.append(label.numpy())

test_labels = np.concatenate(test_labels)

# Calculate confusion matrix
conf_matrix = confusion_matrix(test_labels, test_predictions)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate overall accuracy
overall_accuracy = np.sum(np.diag(conf_matrix)) / np.sum(conf_matrix)
print("Overall Accuracy:", overall_accuracy)

# Calculate precision, recall, and F1-score using classification report
report = classification_report(test_labels, test_predictions, digits=4)
print("Classification Report:")
print(report)
```

```
1/1 [==============================] - 3s 3s/step
Confusion Matrix:
[[10  0]
 [22  0]]
Overall Accuracy: 0.3125
Classification Report:
              precision    recall  f1-score   support

           0     0.3125    1.0000    0.4762        10
           1     0.0000    0.0000    0.0000        22

    accuracy                         0.3125        32
   macro avg     0.1562    0.5000    0.2381        32
weighted avg     0.0977    0.3125    0.1488        32
```

بخش 3: scheduling نرخ یادگیری

قسمت 1:

- **Cosine annealing**

روش (Cosine Annealing) یکی از روش‌های بهینه‌سازی مورد استفاده در آموزش شبکه‌های عصبی عمیق است. این روش به‌طور خاص در بهینه‌سازی نرخ یادگیری (learning rate) مورد استفاده قرار می‌گیرد.

در این روش ، نرخ یادگیری به طور متناسب با کسینوس تغییر می‌کند، به گونه‌ای که در ابتدا نرخ یادگیری بسیار بالا است و سپس به تدریج کاهش پیدا می‌کند. این تغییر نرخ یادگیری به شکل یک (cosine curve) است که از یک مقدار بالا شروع می‌شود و به مقداری کمتر از نقطه اوج می‌رسد، سپس دوباره افزایش پیدا می‌کند و این فرآیند تکرار می‌شود.

در آموزش شبکه‌های عمیق، نرخ یادگیری می‌تواند نقش مهمی در عملکرد و کیفیت آموزش داشته باشد. استفاده از روش کوزین آنیلینگ می‌تواند به کاهش احتمال بیش‌برازش کمک کند. با کاهش نرخ یادگیری در مراحل پایانی آموزش، احتمال دچار شدن به بیش‌برازش کمتر می‌شود و شبکه قادر به یادگیری الگوهای عمومی‌تری می‌شود که به بهبود دقت روی داده‌های آزمون منجر می‌شود. همچنین استفاده از این روش می‌توان به پایدارسازی فرآیند آموزش کمک کند. با تنظیم نرخ یادگیری به طور دقیق با استفاده

از کوزین آنیلینگ، ممکن است نوسانات و نویزهایی که در نمودارهای لاس و دقت مشاهده می‌شوند، کاهش یابد و آموزش به طور پایدارتر ادامه یابد.

برای پیاده سازی این قسمت از کلاس cosine decay در کراس استفاده شد و مدل را دوباره برای 30 ایپاک ترین میکنیم. این دفعه مقدار لرنینگ ریت متغیر است.

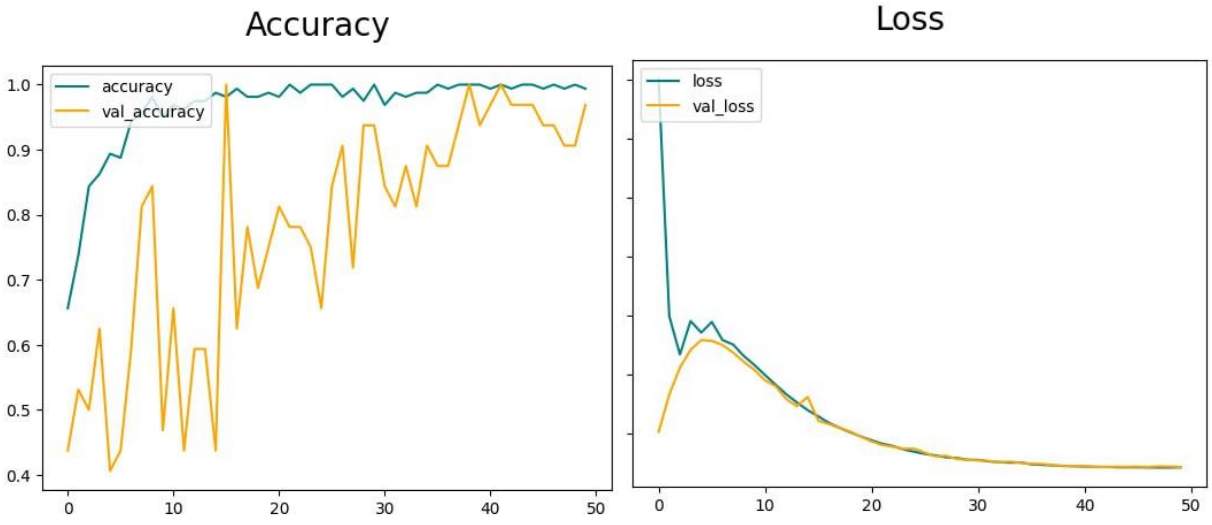
```
from tensorflow.keras.experimental import CosineDecay
from tensorflow.keras.callbacks import LearningRateScheduler

initial_learning_rate = 0.01
cosine_decay = CosineDecay(initial_learning_rate, total_steps=total_steps)

# Define a learning rate scheduler callback
lr_scheduler = LearningRateScheduler(lambda epoch:
cosine_decay(epoch).numpy(), verbose=1)

hist = model.fit(train, epochs=30, validation_data=val,
callbacks=[lr_scheduler])
```

همینطور که مشاهده میشود با استفاده از این کلاس لاس به اندازه قابل ملاحظه ای کمتر شده و دقت بهبود یافته است. مقدار لاس نهایی 0.21ومقدار دقت 0.93 شد.



- **Reduce LR On Plateau**

ReduceLROnPlateau یک کالبک است که در فرآیند آموزش شبکه‌های عصبی استفاده می‌شود تا نرخ یادگیری را به‌طور خودکار کاهش دهد زمانی که یک معیار تعیین‌کننده (مانند لاس یا دقت) دیگر به تغییرات معنی‌داری نیز نرسیده باشد.

این کالبک می‌تواند کمک کند تا فرآیند آموزش شبکه متعادل‌تر و پایدارتر باشد. با کاهش نرخ یادگیری زمانی که آموزش دچار نوسانات یا نویز می‌شود، می‌توان از ایجاد وقفه‌های ناگهانی در آموزش جلوگیری کرد. همچنین با استفاده از این کالبک، می‌توان زمان آموزش شبکه را بهبود بخشید. زمانی که آموزش به نقطه کلیدی می‌رسد و بهتر است نرخ یادگیری را کاهش داد، این کالبک می‌تواند به طور خودکار این کار را انجام دهد. با کاهش نرخ یادگیری زمانی که مدل دچار بیش‌برازش می‌شود، این کالبک می‌تواند به پیشگیری از این مشکل کمک کند و عملکرد مدل را روی داده‌های جدید بهبود بخشد.

به طور خلاصه، ReduceLROnPlateau یک کالبک مهم در آموزش شبکه‌های عصبی است که با کاهش نرخ یادگیری در لحظات مناسب، می‌تواند به پایدارسازی آموزش، جلوگیری از بیش‌برازش و بهبود عملکرد مدل کمک کند.

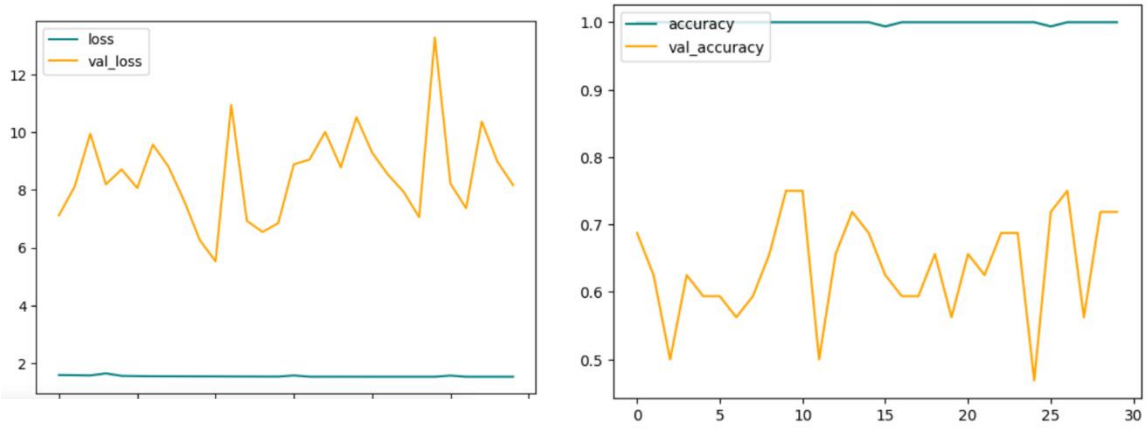
برای این قسمت از کلاس reducelronplateau در کتابخانه کراس استفاده کردیم و مدل را برای 30 ایپاک ترین کردیم.

```
from tensorflow.keras.callbacks import ReduceLROnPlateau

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
min_lr=1e-6)

hist = model.fit(train, epochs=30, validation_data=val,
callbacks=[reduce_lr])
```

همینطور که مشاهده میشود با استفاده از این کلاس لاس به اندازه قابل ملاحظه ای کمتر شده و دقت بهبود یافته است.



قسمت 2:

زمانی که از روش های cosine annealing , reduce lr برای کنترل مقدار نرخ یادگیری در آموزش استفاده می‌شود، تغییراتی در منحنی‌های دقت و لاس دیده می‌شود که به تغییرات در مقدار نرخ یادگیری مربوط است و با نمودار های مشاهده شده در قسمت قبل متفاوت است.

برای مثال زمانی که نرخ یادگیری کاهش می‌یابد، منحنی دقت ممکن است با تغییر نرخ یادگیری تغییراتی نسبت به حالتی که از یک نرخ یادگیری ثابت استفاده می‌شود، نشان دهد. معمولاً در ابتدا با کاهش نرخ یادگیری، دقت ممکن است کمی کاهش یابد، اما سپس با پایدار شدن فرآیند آموزش و یادگیری الگوهای بهتر، دقت می‌تواند بهبود یابد.

تغییرات در نرخ یادگیری می‌تواند تأثیر زیادی بر روی منحنی لاس داشته باشد. هنگامی که نرخ یادگیری کاهش می‌یابد، ممکن است منحنی لاس نیز تغییر کند. این تغییر ممکن است با کاهش سرعت یادگیری در مراحل آخر آموزش، رویایی از بهبود در عملکرد شبکه بدست آورد.

قسمت 3:

در این قسمت کد مربوط به معیارهای ارزیابی که در قسمت های قبل توضیح داده شد را برای بلاک های اضافه شده دوباره اجرا میکنیم و مشاهده میکنیم که به صورت زیر نتیجه میدهد:

```
1/1 [==============================] - 2s 2s/step
Confusion Matrix:
[[ 9  0]
 [23  0]]
Overall Accuracy: 0.28125
Classification Report:
              precision    recall  f1-score   support

           0     0.2812    1.0000    0.4390         9
           1     0.0000    0.0000    0.0000        23

    accuracy                         0.2812        32
   macro avg     0.1406    0.5000    0.2195        32
weighted avg     0.0791    0.2812    0.1235        32
```

```
1/1 [==============================] - 2s 2s/step
Confusion Matrix:
[[11  0]
 [21  0]]
Overall Accuracy: 0.34375
Classification Report:
              precision    recall  f1-score   support

           0     0.3438    1.0000    0.5116        11
           1     0.0000    0.0000    0.0000        21

    accuracy                         0.3438        32
   macro avg     0.1719    0.5000    0.2558        32
weighted avg     0.1182    0.3438    0.1759        32
```

بخش 4: Regularization

قسمت 1:

ابتدا earlystopping را import می کنیم. سپس EarlyStopping callback را برای نظارت بر (val_loss) تعریف می کنیم و در صورت عدم بهبودی برای 3 دوره متوالی آموزش را متوقف می کنیم(patience=3).

```python
from tensorflow.keras.callbacks import EarlyStopping

# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1, restore_best_weights=True)

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

# Train the model with early stopping callback
hist = model.fit(train, epochs=30, validation_data=val, callbacks=[early_stopping])

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

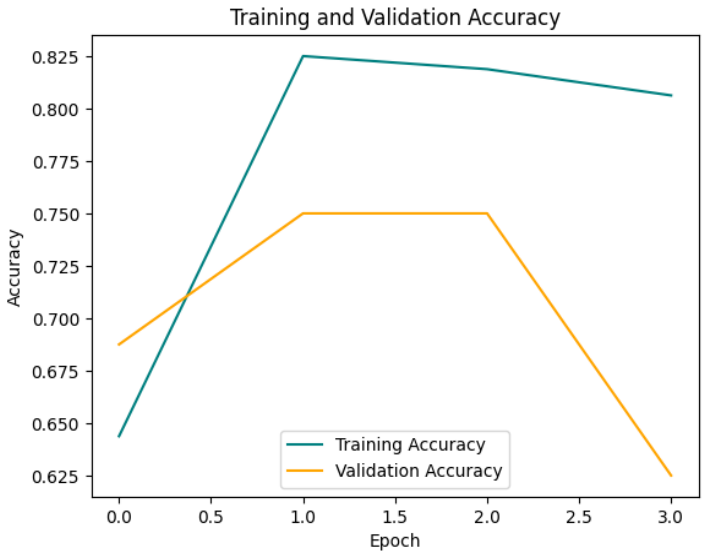
که خروجی به این صورت می باشد:

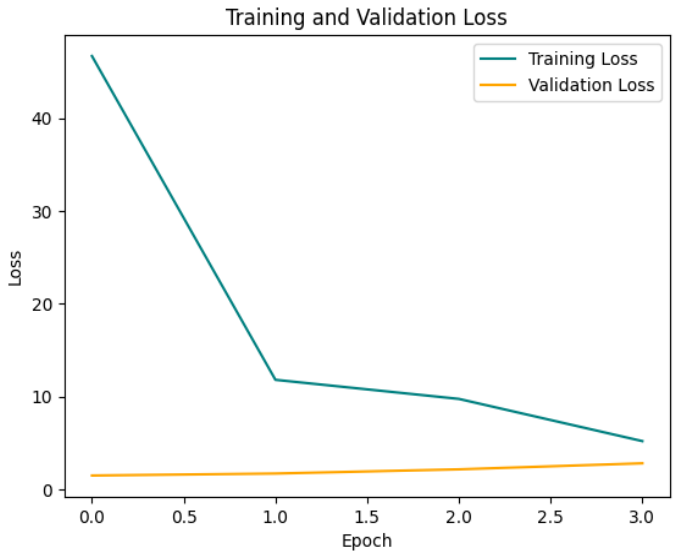
```
Epoch 1/30
5/5 [==============================] - 30s 5s/step - loss: 46.6905 - accuracy: 0.6438 - val_loss: 1.4956 - val_
Epoch 2/30
5/5 [==============================] - 24s 5s/step - loss: 11.8072 - accuracy: 0.8250 - val_loss: 1.7092 - val_
Epoch 3/30
5/5 [==============================] - 30s 5s/step - loss: 9.7500 - accuracy: 0.8188 - val_loss: 2.1617 - val_a
Epoch 4/30
5/5 [==============================] - ETA: 0s - loss: 5.2010 - accuracy: 0.8062Restoring model weights from th
5/5 [==============================] - 24s 5s/step - loss: 5.2010 - accuracy: 0.8062 - val_loss: 2.8119 - val_a
Epoch 4: early stopping
1/1 [==============================] - 1s 1s/step - loss: 1.6571 - accuracy: 0.5938
Test Loss: 1.6570868492126465
Test Accuracy: 0.59375
```

سپس confusion matrix را محاسبه می کنیم:

```
1/1 [==============================] - 2s 2s/step
Confusion Matrix:
[[ 8  7]
 [ 5 12]]
```

نمودار های loss و accuracy :





که در اینجا می توان مشاهده کرد که از 3 epoch به بعد مدل overfit می شود پس early stopping اتفاق می افتد می مدل دیگر به یادگیری ادامه نمی دهد

قسمت 2:

طبق مدلی که قبلا داشتیم به آن L2 Regularization اضافه می کنیم، L2 regularization روی دومین لایه separable conv و اولین لایه FC متصل میشود برای جلوگیری از overfitting.

مدل به این صورت می شود:

```
# Define the model
model = Sequential()
model.add(SeparableConv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(SeparableConv2D(64, kernel_size=(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.001)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(1, activation='sigmoid'))
```

سپس روی دادگان تست ارزیابی می کنیم:

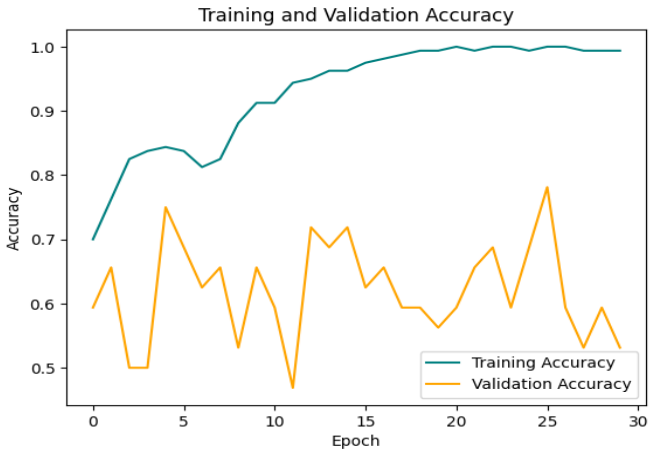
```
1/1 [==============================] - 3s 3s/step - loss: 9.5735 - accuracy: 0.6250
Test Loss: 9.573481559753418
Test Accuracy: 0.625
1/1 [==============================] - 1s 1s/step
```

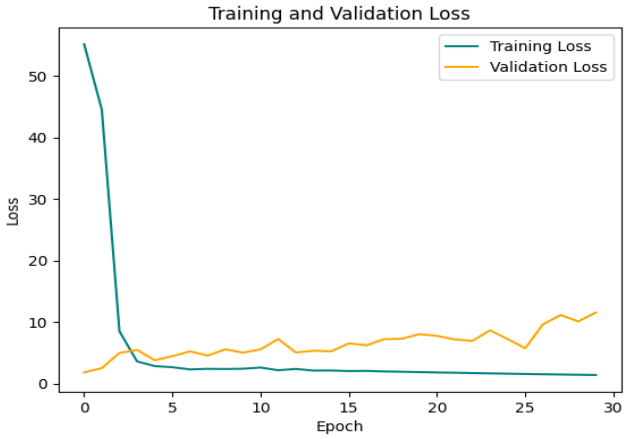
سپس confusion matrix را محاسبه می کنیم:

```
Confusion Matrix:
[[11  0]
 [21  0]]
```

تکنیک‌های regularization با استفاده از منظم سازی، می توانید انتظار داشته باشید که مدل عملکرد تعمیم بهتری داشته باشد، به خصوص در داده های دیده نشده، مانند پنالتی L2، با جریمه کردن وزن‌های بزرگ در مدل، به جلوگیری از overfitting کمک می‌کند. با استفاده از regularization ، می توانید انتظار داشته باشید که مدل عملکرد تعمیم بهتری داشته باشد، به خصوص در داده های دیده نشده. ماتریس confusion و دقت کلی در مجموعه آزمایشی باید عملکرد بهبود مدل regularizedرا در مقایسه با مدل قبلی نشان دهند. که همان طور که مشاهده می کنیم با اضافه کردن regularization دقت مدل روی داده های تست بیشتر از قسمت قبل شده است.(از 59 به 62.5 درصد رسیده است)

نمودار های loss و accuracy :





قسمت 3:

در این قسمت به لایه های مدل dropout نیز اضافه می شود. dropout با کاهش وابستگی به نورون‌های خاص در طول یادگیری، به جلوگیری overfitting کمک می‌کند. این شبکه را تشویق می‌کند تا ویژگی‌های قوی‌تری را بیاموزد و عملکرد تعمیم آن را روی داده‌های دیده نشده بهبود می‌بخشد.

```
# Define the model
model = Sequential()
model.add(SeparableConv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(SeparableConv2D(64, kernel_size=(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.001)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

ارزیابی روی دادگان تست:

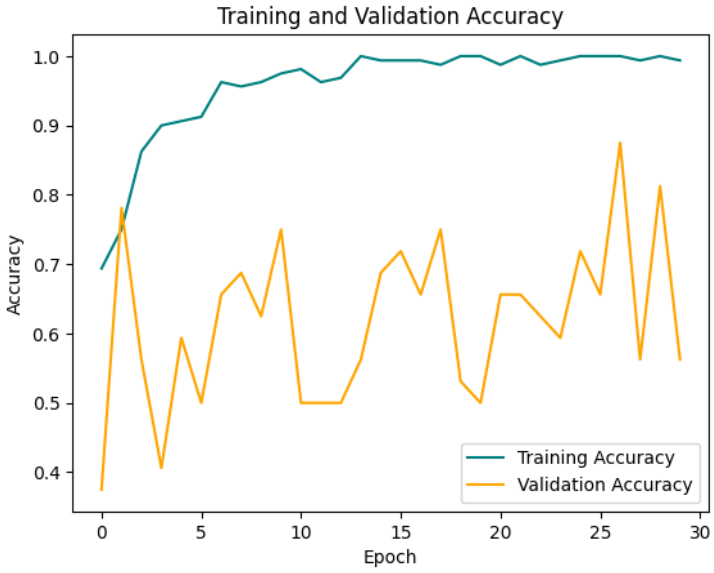
```
1/1 [==============================] - 1s 1s/step - loss: 5.0536 - accuracy: 0.7500
Test Loss: 5.053645133972168
Test Accuracy: 0.75
1/1 [==============================] - 1s 1s/step
```

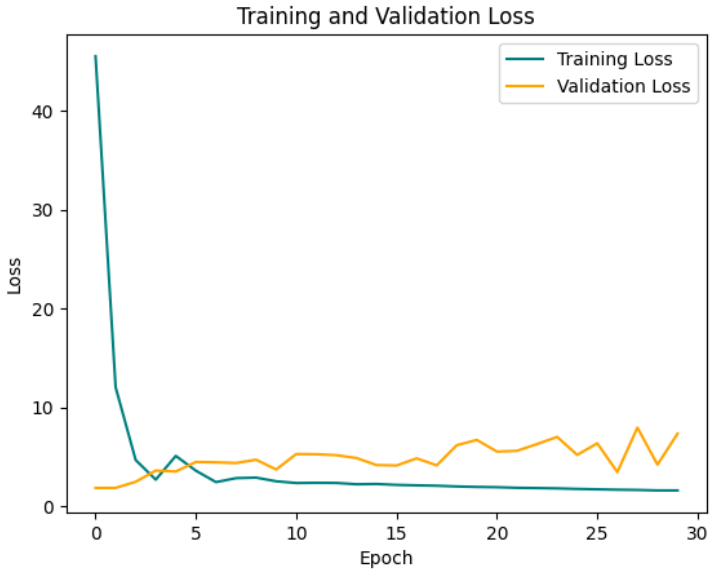
همان طور که می توان مشاهده کرد طبق انتظاری که می رفت می بایست با اضافه کردن dropout دقت مدل از 62.5 به 75 درصد افزایش پیدا کرد که این به این معنا است که مدل بهتر آموزش دیده است.

سپس confusion matrix را محاسبه می کنیم:

```
Confusion Matrix:
[[ 0 15]
 [ 0 17]]
```

نمودار های loss و accuracy :





بخش 5: یادگیری انتقالی

ResNet:

قسمت 1:

با استفاده از دستورهای زیر و با استفاده از کتابخانه ها مدل ها را به جز لایه نهایی بارگذاری کردیم.

```
import tensorflow as tf
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import Adam

resnet_model = tf.keras.applications.ResNet50(weights='imagenet',
include_top=False)


for layer in resnet_model.layers:
    layer.trainable = False
```

```
resnet_model.summary()
```

قسمت 2:

با استفاده از دستورهای زیر لایه طبقه بندی مناسب را به مدل اضافه کردیم.

```
x = GlobalAveragePooling2D()(resnet_model.output)
x = Dense(256, activation='relu')(x)
output = Dense(1, activation='softmax')(x)

model_resnet = Model(inputs=resnet_model.input, outputs=output)
```

قسمت 3:

مدل را برای 30 ایپاک آموزش دادیم و نتایج زیر به دست آمد.

```
model_resnet.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])

history_resnet = model_resnet.fit(train, epochs=30, validation_data=val)
```

```
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
resnet_predictions = model_resnet.predict(test)
resnet_pred_labels = np.argmax(resnet_predictions, axis=1)
resnet_true_labels = np.argmax(test, axis=1)

resnet_accuracy = accuracy_score(resnet_true_labels, resnet_pred_labels)

resnet_confusion = confusion_matrix(resnet_true_labels,
resnet_pred_labels)

print("ResNet-18 Accuracy:", resnet_accuracy)
print("ResNet-18 Confusion Matrix:")
print(resnet_confusion)
```

SqueezeNet :

با استفاده از دستورهاتی که در قسمت قبل و با استفاده از کتابخانه ها مدل ها امکان پذیر نمی باشد زیرا هیچ مدل SqueezeNet به طور مستقیم در ماژول tensorflow.keras.applications موجود

نیست . با این حال، می‌توانیم مدل SqueezeNet را با استفاده از روش‌های دیگری load کنیم، مانند load کردن وزن‌ها به صورت دستی یا پیاده‌سازی معماری مدل خود. به این ترتیب ناچار هستیم به صورت دستی آن را بنویسیم.

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, concatenate, Dropout, Flatten, Dense
from tensorflow.keras.models import Model

def fire_module(x, squeeze, expand):
    squeezed = Conv2D(squeeze, (1, 1), activation='relu', padding='same')(x)
    expanded1x1 = Conv2D(expand, (1, 1), activation='relu', padding='same')(squeezed)
    expanded3x3 = Conv2D(expand, (3, 3), activation='relu', padding='same')(squeezed)
    output = concatenate([expanded1x1, expanded3x3], axis=-1)
    return output
```

سپس تابع squeezenet :

```
def SqueezeNet(input_shape=(224, 224, 3), num_classes=1000):
    input_img = Input(shape=input_shape)
    x = Conv2D(96, (7, 7), strides=(2, 2), activation='relu', padding='same')(input_img)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    x = fire_module(x, squeeze=16, expand=64)
    x = fire_module(x, squeeze=16, expand=64)
    x = fire_module(x, squeeze=32, expand=128)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    x = fire_module(x, squeeze=32, expand=128)
    x = fire_module(x, squeeze=48, expand=192)
    x = fire_module(x, squeeze=48, expand=192)
    x = fire_module(x, squeeze=64, expand=256)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    x = fire_module(x, squeeze=64, expand=256)
    x = Dropout(0.5)(x)

    x = Conv2D(num_classes, (1, 1), activation='relu', padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=input_img, outputs=output)
    return model
```

در نهایت:

```
num_classes = 100

# Load SqueezeNet
transfer_model = SqueezeNet(input_shape=(224, 224, 3), num_classes=num_classes)

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

transfer_model.summary()
```