Jellyfish

CCDS 412 - Applied Deep Learning

Final Project: Phase 2

Contents

Objective

Using Deep Learning techniques to classify jellyfish images into different categories and Identify the species of jellyfish based on their physical characteristics

DESCRIPTION OF DATA

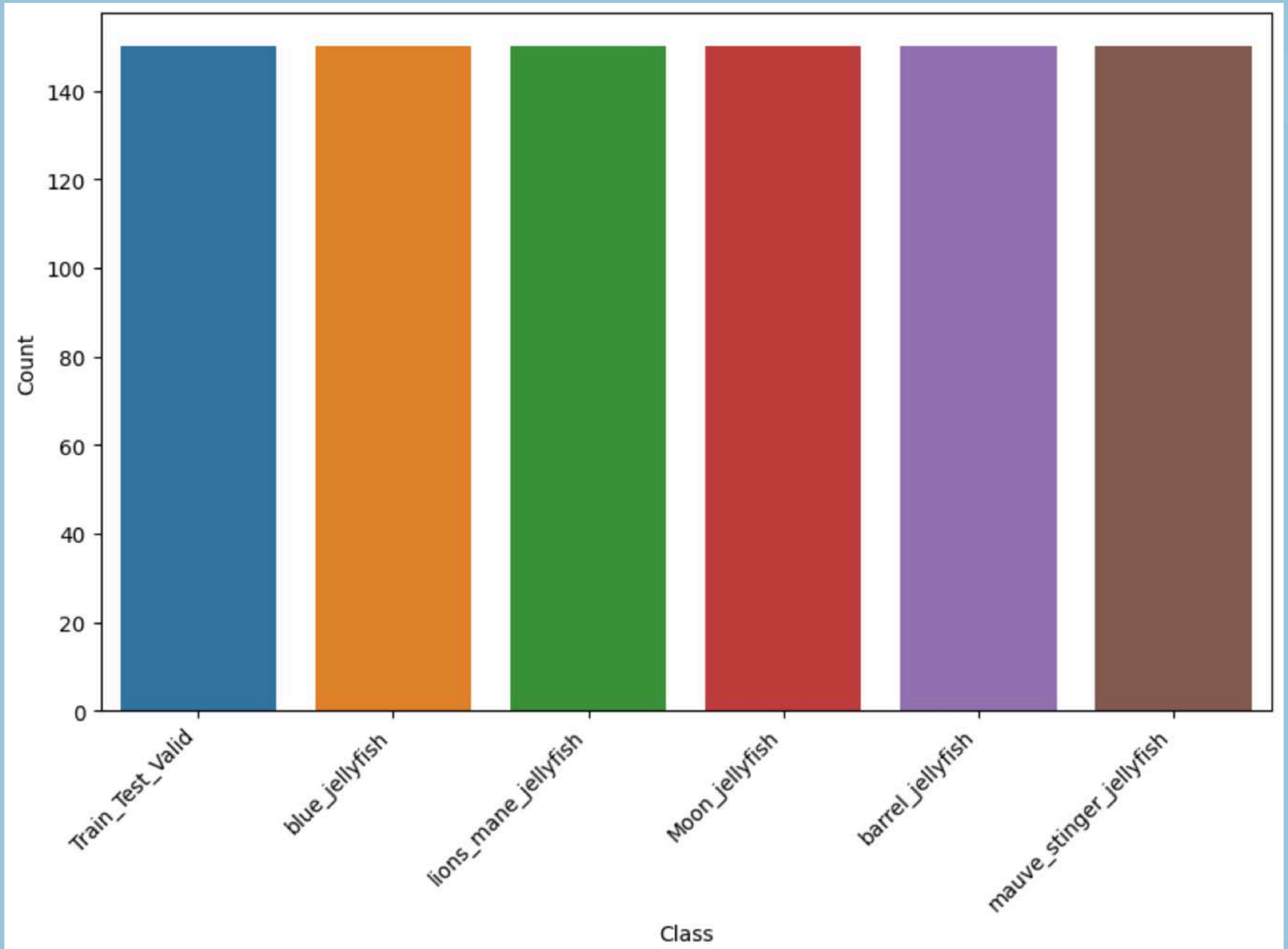
Description of Data

The dataset contains 900 images of jellyfish belonging to six different categories and species:

1. **Mauve stinger jellyfish**
2. **Moon jellyfish**
3. **Barrel jellyfish**
4. **Blue jellyfish**
5. **Compass jellyfish**
6. **Lion’s mane jellyfish**

We will apply DL techniques to:
gain insights into jellyfish classification, species identification, and color analysis.

Description of Data



Description of Data

Show some of the image

OUR PROBLEM

Our Problem

Our project aims to classify images of jellyfish into different ones and identify jellyfish types based on their physical characteristics. Therefore, the problem boil down to:

- **the difficulty of photographing a living organism underwater:** there are not enough images to train the model, so we will need to increase the number of images that are different in size and modify and standardize the sizes.
- **some of the images are unclear:** jellyfish have colors similar to coral reefs and interfere with us, so we need to enlarge the image to clarify the details so that the model can train.

PRE-PROCESSING

Data Augmentation

```
datagen = ImageDataGenerator(
    rotation_range=30,          # Randomly rotate images by 20 degrees
    #brightness_range=[0., 0.5],  # Adjust brightness between 10% and 200%
    width_shift_range=0.1,      # Randomly shift images horizontally by 20% of the width
    height_shift_range=0.1,     # Randomly shift images vertically by 20% of the height
    zoom_range=0.3,             # Randomly zoom into images
    horizontal_flip=True,       # Randomly flip images horizontally
    vertical_flip=True,         # Randomly flip images vertically
)
datagen1 = ImageDataGenerator()

train_generator = datagen.flow(X_train, y_train, batch_size=20)
val_generator = datagen1.flow(X_test, y_test, batch_size=20)
```

Data Augmentation

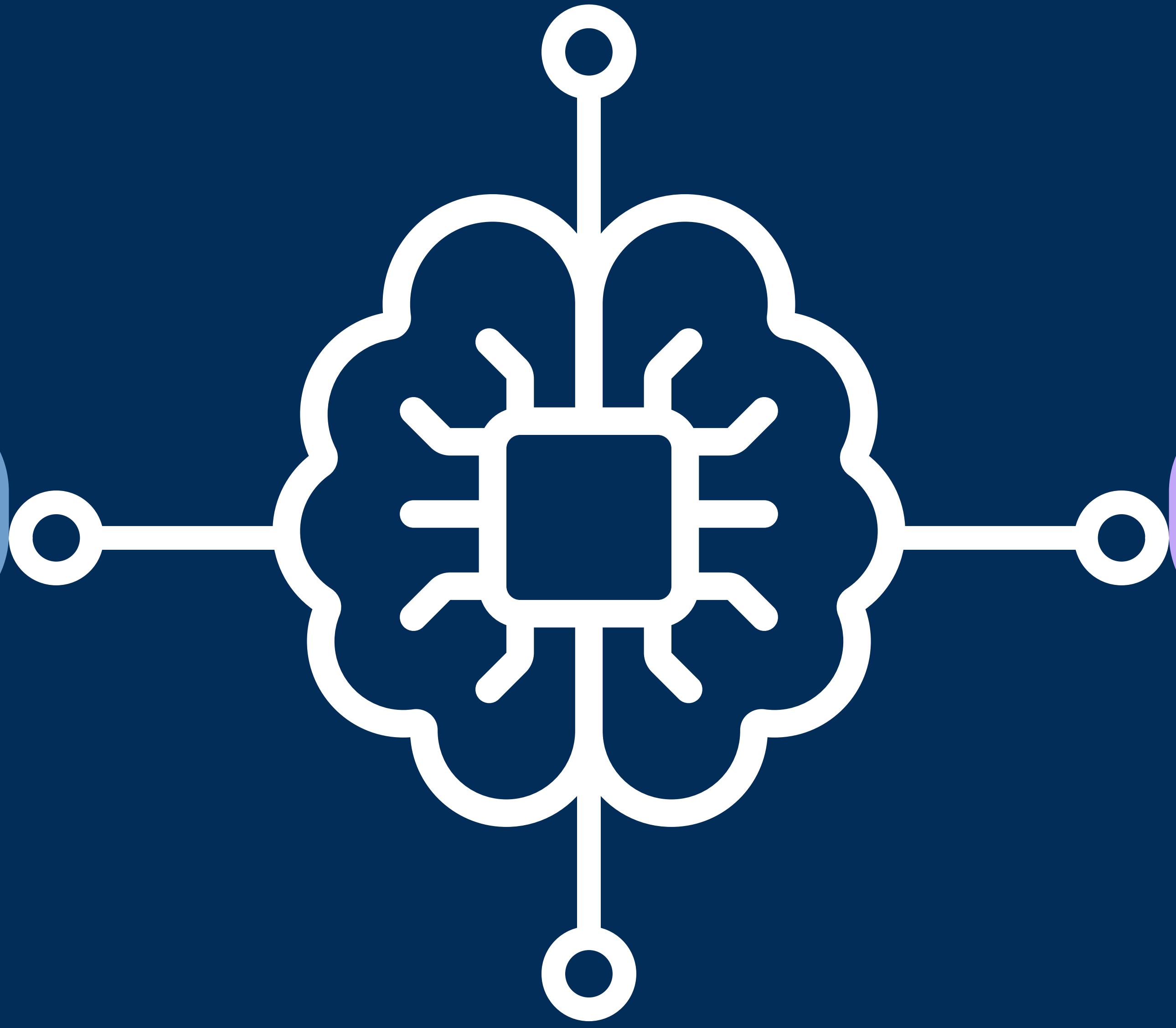
Original Image

Augmented Image

OUR MODELS

OUR MODELS

CNN Model



VGGNet Model

AlexNet Model

ConvNet Model

Phase 1

CNN Model

Define the Model

```
from keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes=6)
y_test = to_categorical(y_test, num_classes=6)
```

```
model.fit(X_train, y_train, validation_data=(X_test,y_test), batch_size=500,epochs=20)
```

Phase 1

CNN Model

Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 16)      208

 max_pooling2d (MaxPooling2  (None, 112, 112, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      2080

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        8256

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 dropout (Dropout)           (None, 28, 28, 64)        0

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 255)               12795135

 dropout_1 (Dropout)         (None, 255)               0

 dense_1 (Dense)             (None, 6)                 1536

=================================================================
Total params: 12807215 (48.86 MB)
Trainable params: 12807215 (48.86 MB)
Non-trainable params: 0 (0.00 Byte)
```

Phase 1

CNN Model

Accuracy

```
Epoch 20/20
2/2 [==============================] - 19s 8s/step - loss: 0.1050 - accuracy: 0.9764 - val_loss: 0.8626 - val_accuracy: 0.7278
```

```
y_hat = model.predict(X_test)
test = model.evaluate(X_test, y_test)
print('Test Loss = ', test[0], 'Test Accuracy = ', test[1])
```

```
6/6 [==============================] - 1s 197ms/step
6/6 [==============================] - 1s 185ms/step - loss: 1.0247 - accuracy: 0.6889
Test Loss =  1.0247085094451904 Test Accuracy =  0.6888889074325562
```

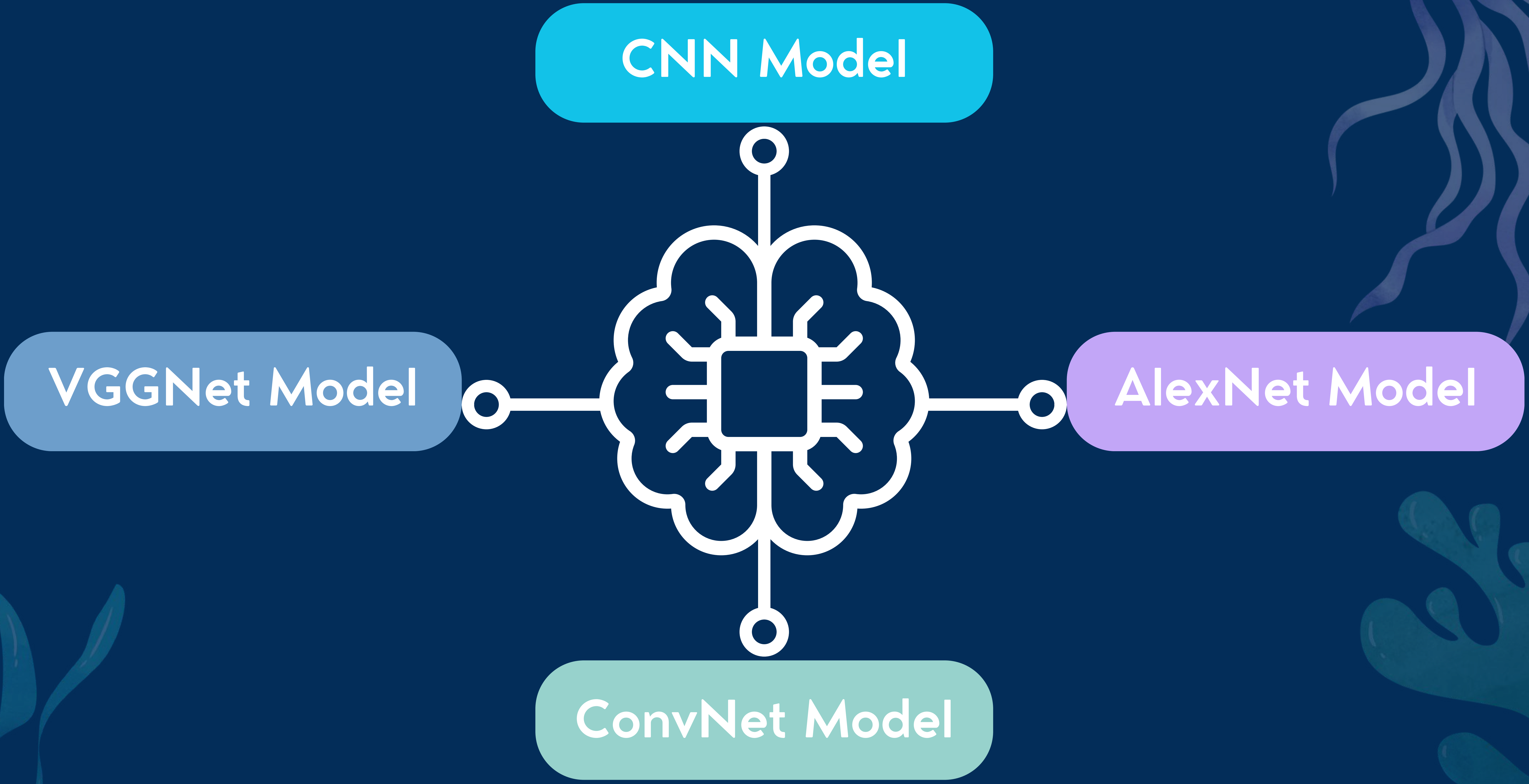
97% - 0.10

Training Accuracy - Loss

68% - 1.02

Testing Accuracy - Loss

OUR MODELS



AlexNet Model

Define the Model

```
# Define and compile AlexNet model
alexnet_model = create_alexnet_model(input_shape=(224, 224, 3), num_classes=6)
alexnet_model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train AlexNet model
alexnet_history = alexnet_model.fit(X_train, y_train, epochs=20, batch_size=128, validation_data=(X_test, y_test))
```

AlexNet Model

Model Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 224, 224, 16)      208

 max_pooling2d_3 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 112, 112, 32)      2080

 max_pooling2d_4 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 56, 56, 64)        8256

 max_pooling2d_5 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 dropout_2 (Dropout)         (None, 28, 28, 64)        0

 flatten_1 (Flatten)         (None, 50176)             0

 dense_2 (Dense)             (None, 255)               12795135

 dropout_3 (Dropout)         (None, 255)               0

 dense_3 (Dense)             (None, 6)                 1536

=================================================================
Total params: 12807215 (48.86 MB)
Trainable params: 12807215 (48.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

AlexNet Model

Accuracy

```
Best Accuracy: 0.6306 at Epoch 20
Best Validation Accuracy: 0.5667 at Epoch 19
```

```
# Evaluate the model on the test data
# Print the test loss and test accuracy
y_hat = alexnet_model.predict(X_test)
test = alexnet_model.evaluate(X_test, y_test)
print('Test Loss = ', test[0], 'Test Accuracy = ', test[1])
```

```
6/6 [==============================] - 3s 506ms/step
6/6 [==============================] - 4s 614ms/step - loss: 1.1495 - accuracy: 0.5556
Test Loss =  1.1494736671447754 Test Accuracy =  0.5555555820465088
```

63% - 0.91

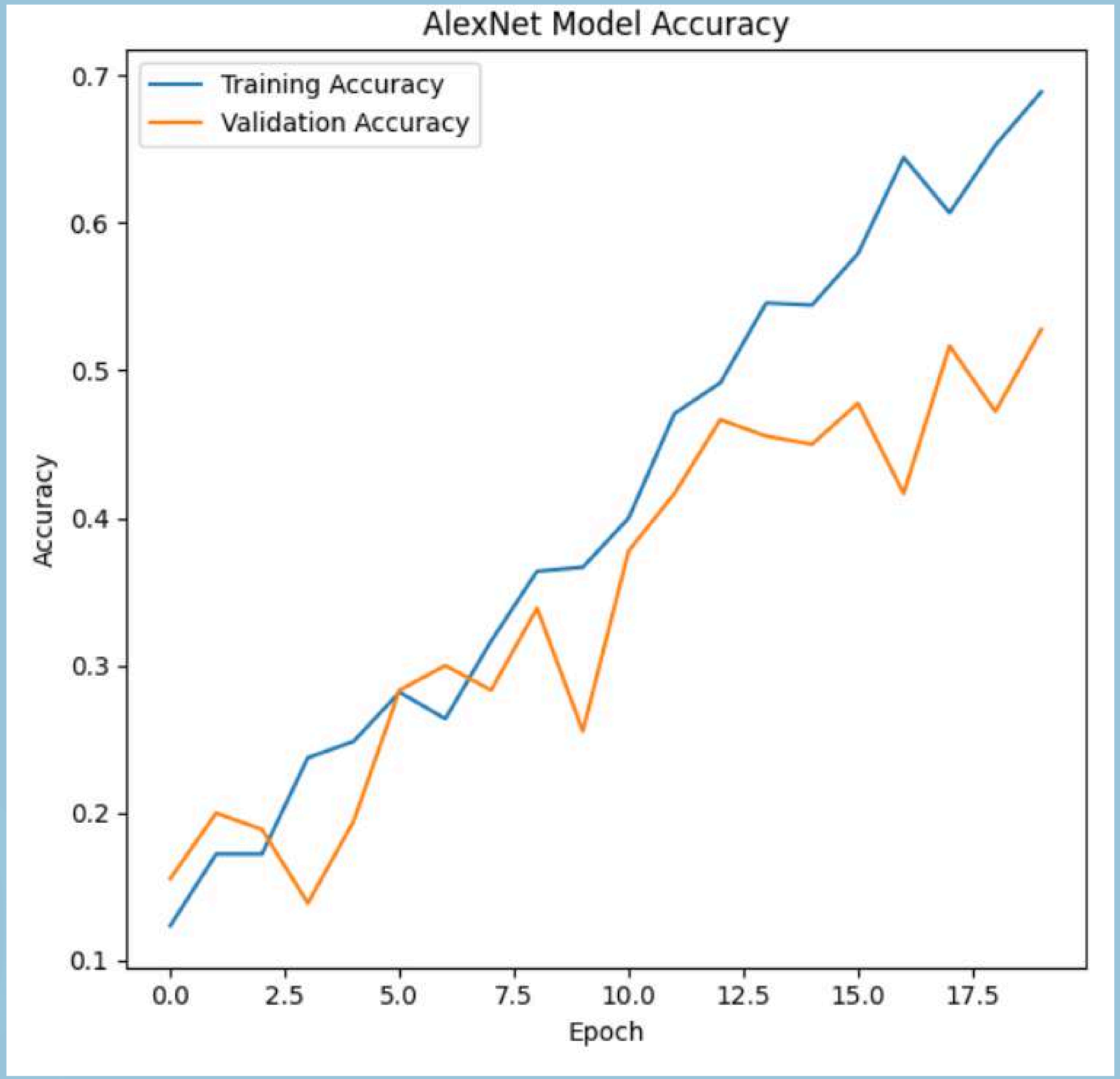
Training Accuracy - Loss

55% - 1.14

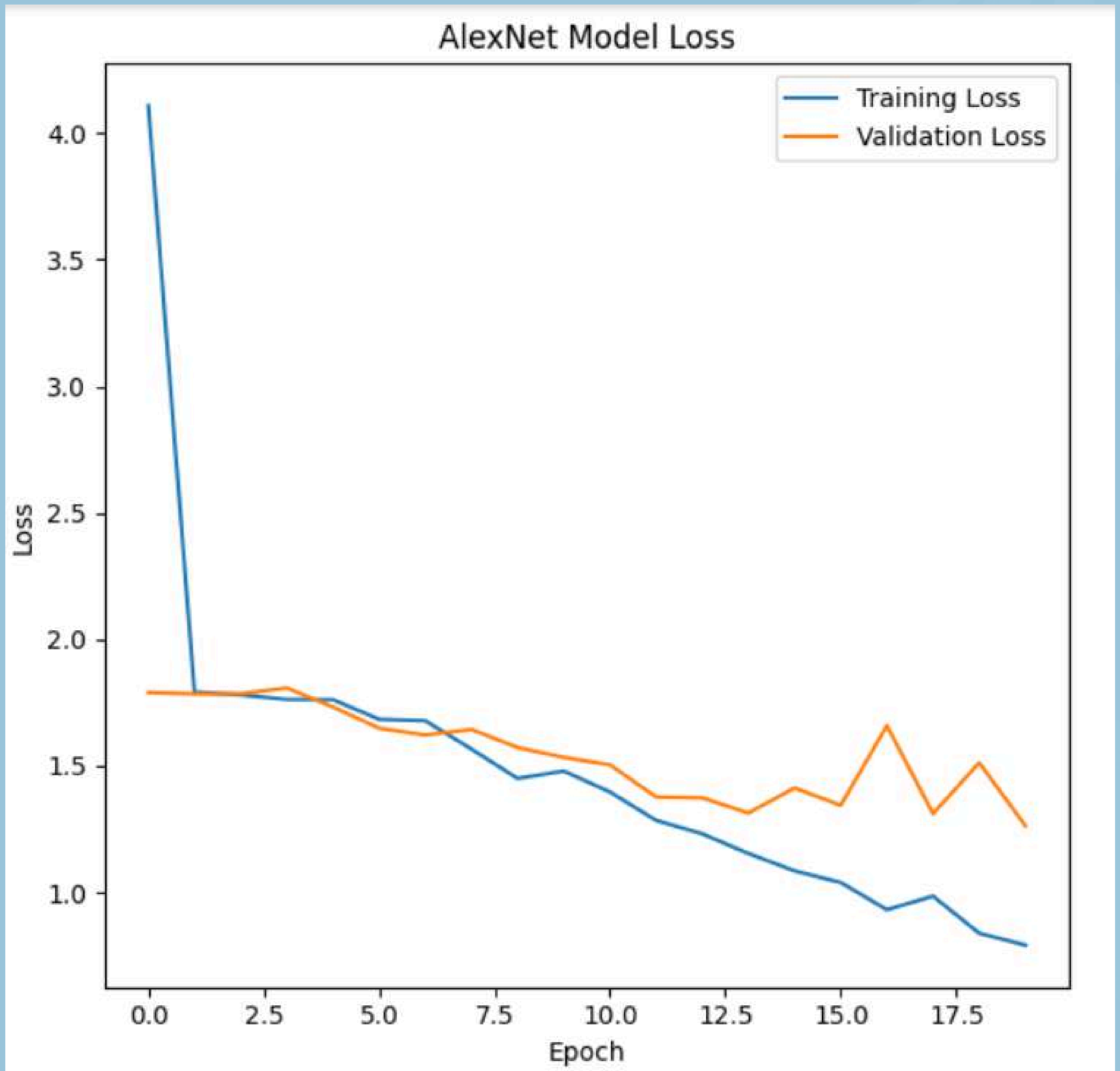
Testing Accuracy - Loss

AlexNet Model

Model Accuracy

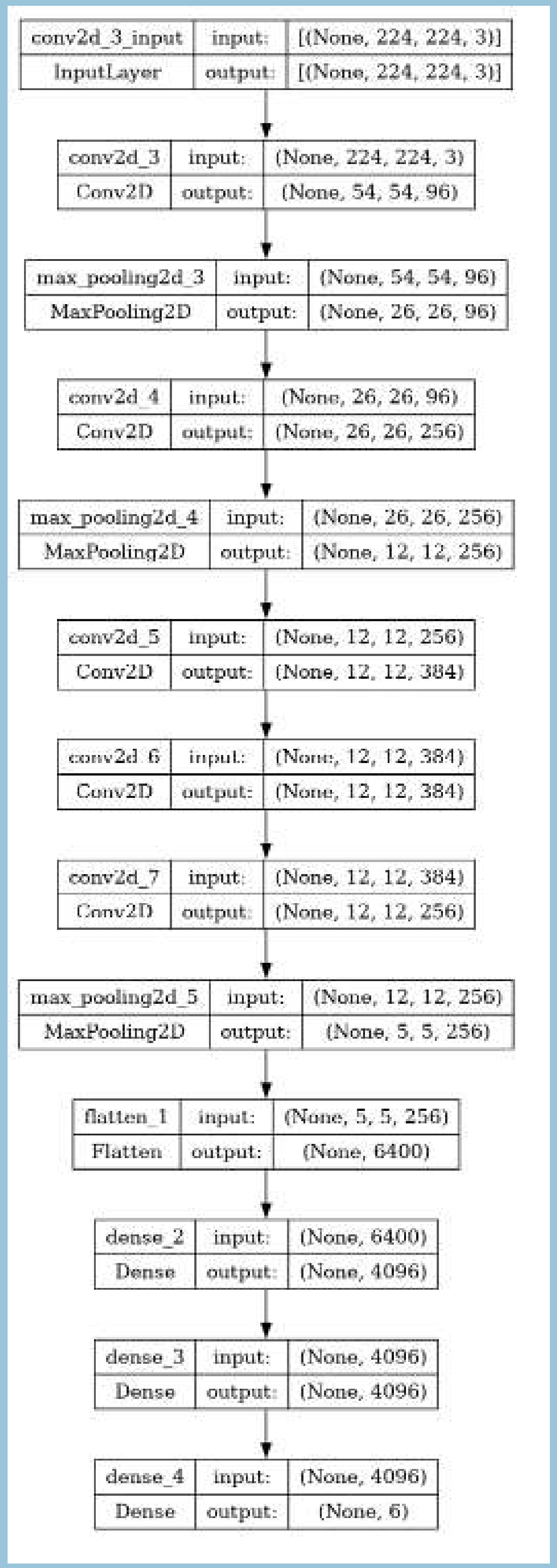


Model Loss

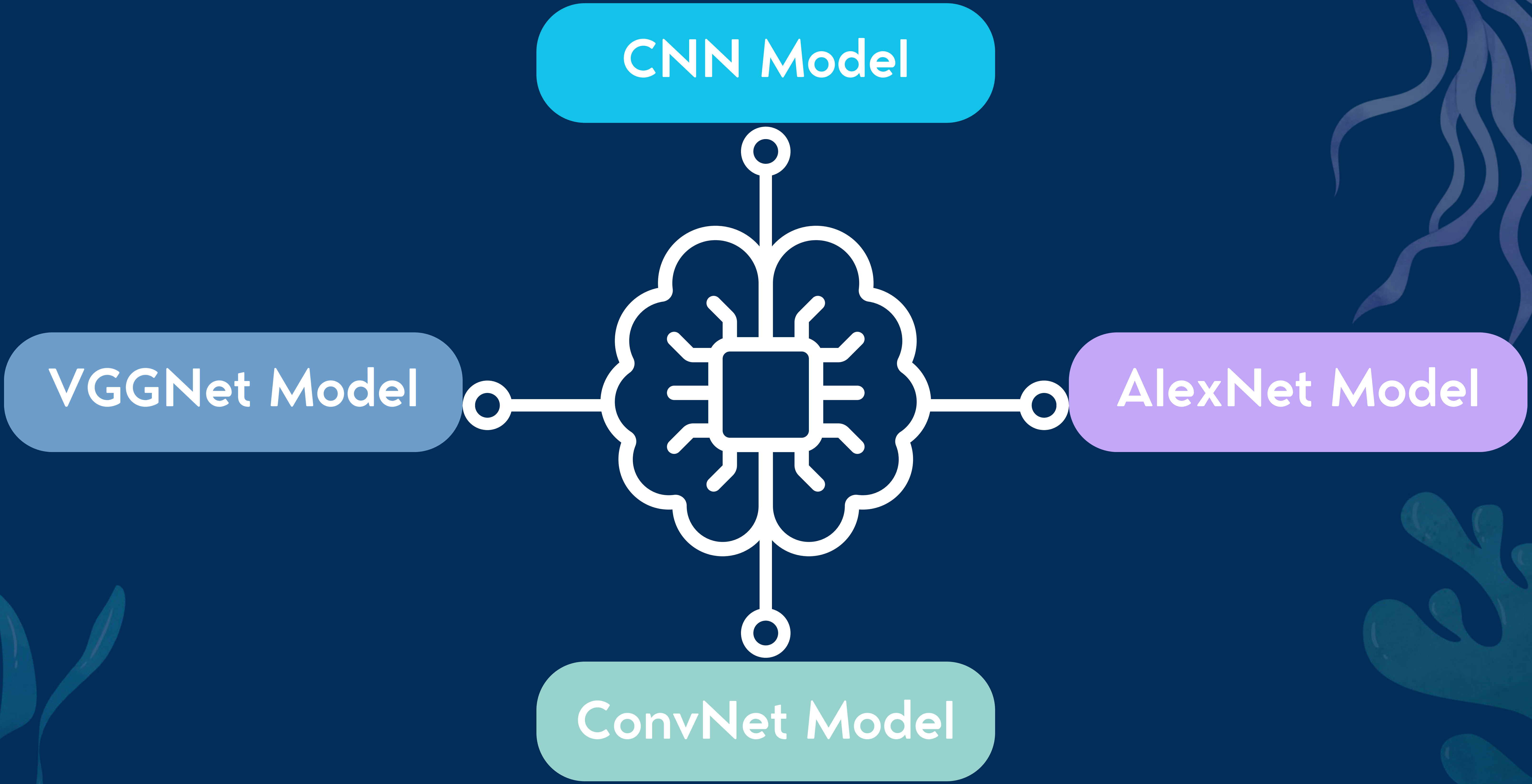


AlexNet Model

Architecture Diagram



OUR MODELS



ConvNet Model

Define the Model

```
# Define and compile ConvNet Tiny model
convnet_tiny_model = create_convnet_tiny_model(input_shape=(224, 224, 3), num_classes=6)
convnet_tiny_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train ConvNet Tiny model
convnet_tiny_history = convnet_tiny_model.fit(X_train, y_train, epochs=15, batch_size=128, validation_data=(X_test, y_test))
```

ConvNet Model

Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 16)      208

 max_pooling2d (MaxPooling2  (None, 112, 112, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      2080

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        8256

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 dropout (Dropout)           (None, 28, 28, 64)        0

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 255)               12795135

 dropout_1 (Dropout)         (None, 255)               0

 dense_1 (Dense)             (None, 6)                 1536

=================================================================
Total params: 12807215 (48.86 MB)
Trainable params: 12807215 (48.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

ConvNet Model

Accuracy

Overfitting

```
Best Accuracy for ConvNet Tiny: 0.9889 at Epoch 15
Best Validation Accuracy for ConvNet Tiny: 0.6667 at Epoch 9
```

```
# Print the test loss and test accuracy
y_hat = convnet_tiny_model.predict(X_test)
test = convnet_tiny_model.evaluate(X_test, y_test)
print('Test Loss = ', test[0], 'Test Accuracy = ', test[1])
```

```
6/6 [==============================] - 3s 405ms/step
6/6 [==============================] - 3s 410ms/step - loss: 1.5764 - accuracy: 0.6222
Test Loss =  1.576395034790039 Test Accuracy =  0.6222222447395325
```

98% - 0.08

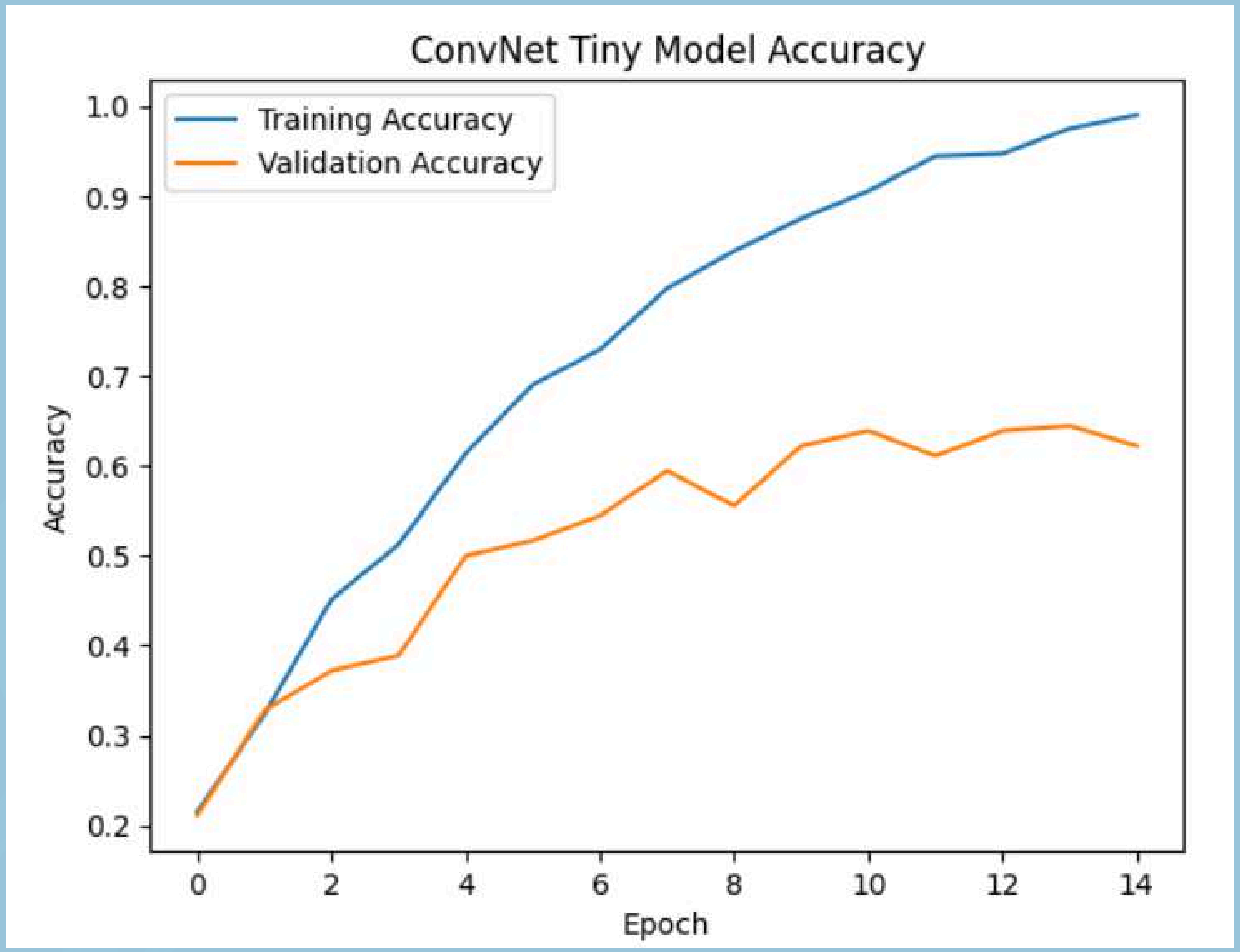
Training Accuracy - Loss

62% - 1.57

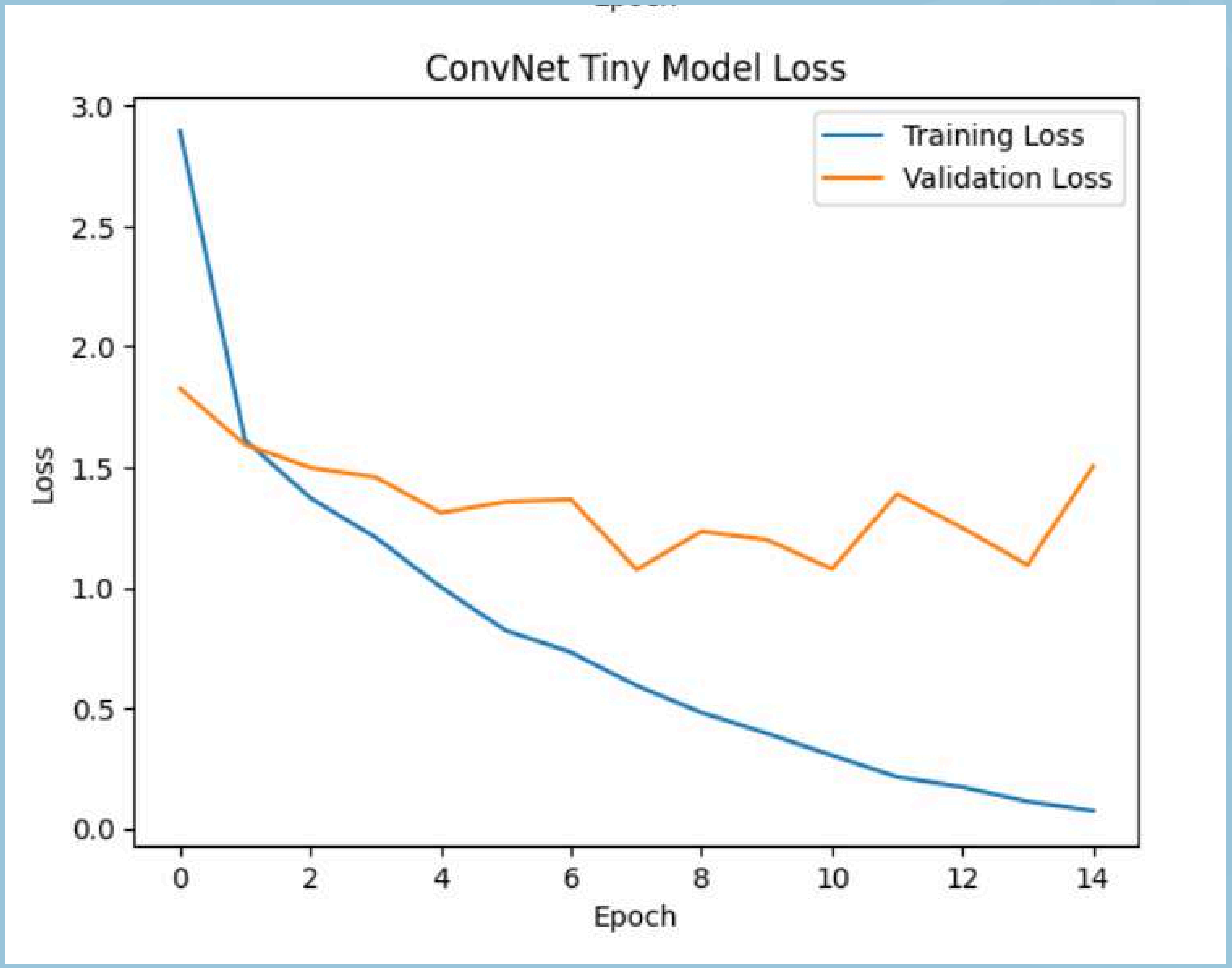
Testing Accuracy - Loss

ConvNet Model

Model Accuracy

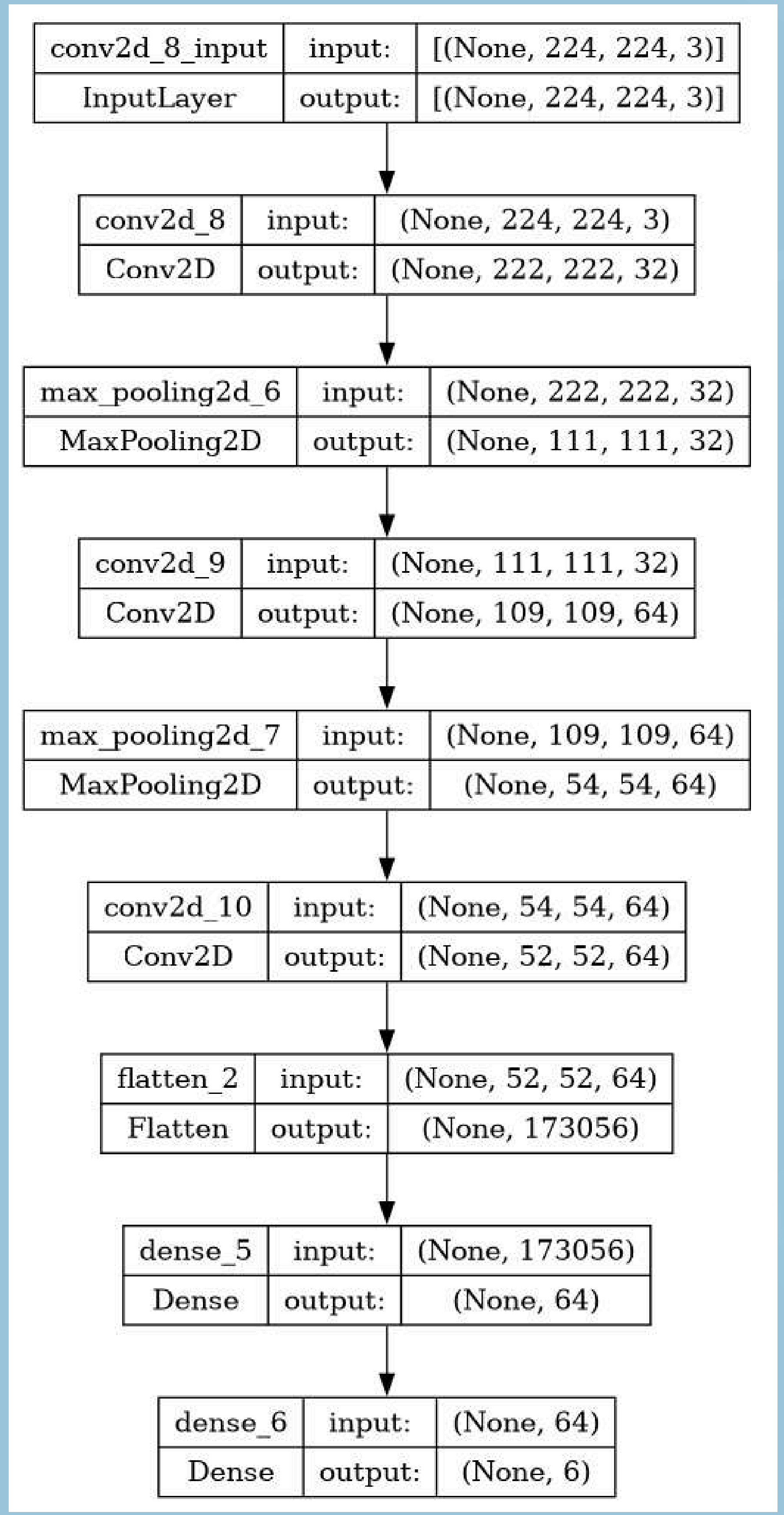


Model Loss

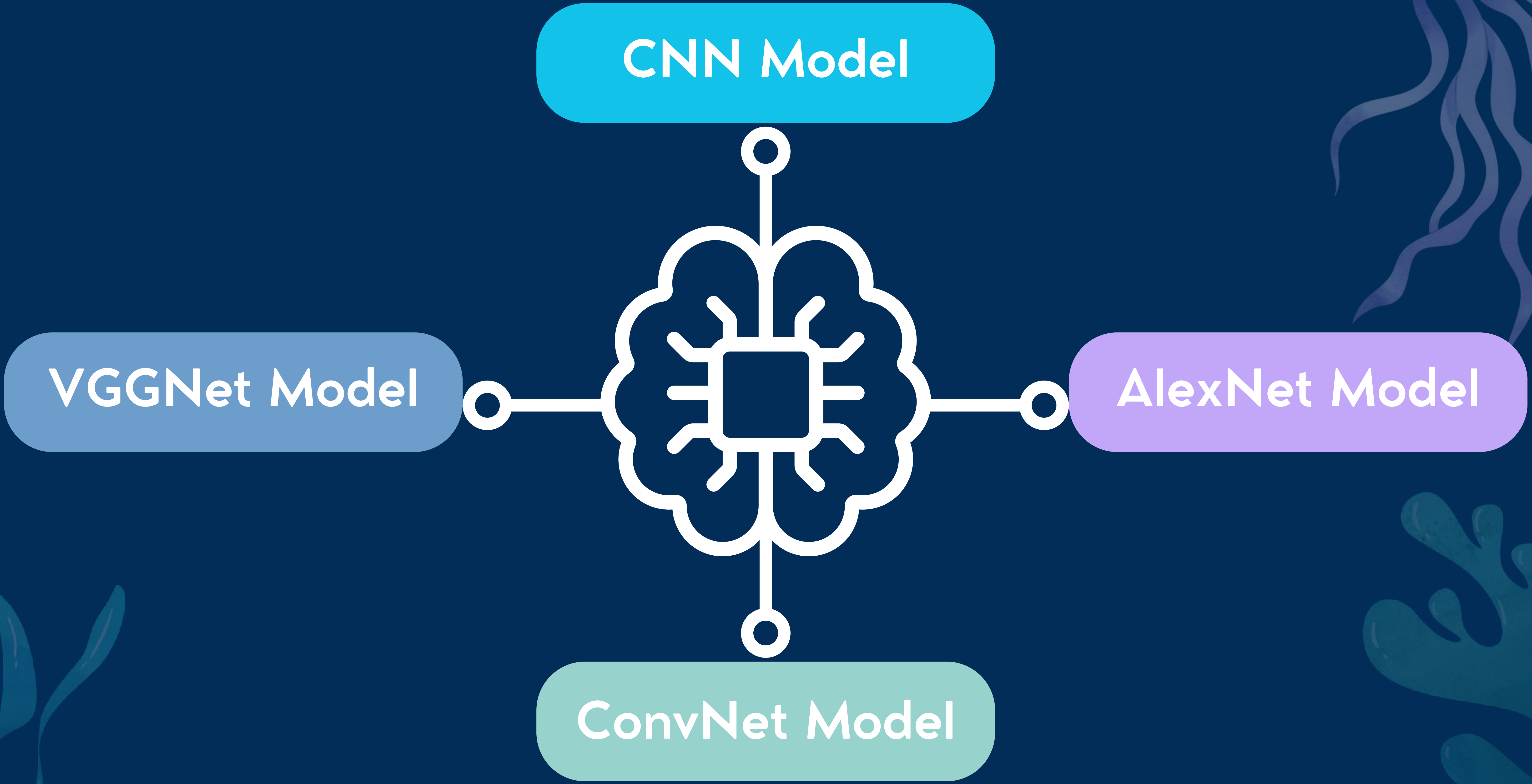


ConvNet Model

Architecture Diagram



OUR MODELS



VGGNet Model

Define the Model

```
# Define another VGGNet model
def create_another_vggnet_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    return model

# Define the input shape and number of classes
input_shape = (224, 224, 3)
num_classes = 6

# Create and compile another VGGNet model
another_vggnet_model = create_another_vggnet_model(input_shape, num_classes)
another_vggnet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
another_vggnet_history = another_vggnet_model.fit(X_train, y_train, epochs=15, batch_size=128, validation_data=(X_test, y_test))
```

VGGNet Model

Model Summary

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 222, 222, 32)      896

 max_pooling2d_8 (MaxPoolin  (None, 111, 111, 32)      0
 g2D)

 conv2d_12 (Conv2D)          (None, 109, 109, 64)      18496

 max_pooling2d_9 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_13 (Conv2D)          (None, 52, 52, 128)       73856

 max_pooling2d_10 (MaxPooli  (None, 26, 26, 128)       0
 ng2D)

 flatten_3 (Flatten)         (None, 86528)             0

 dense_7 (Dense)             (None, 256)               22151424

 dense_8 (Dense)             (None, 6)                 1542

=================================================================
Total params: 22246214 (84.86 MB)
Trainable params: 22246214 (84.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

VGGNet Model

Accuracy

Overfitting

```
Best Accuracy: 0.9500 at Epoch 15
Best Validation Accuracy: 0.5667 at Epoch 15
```

```python
# Print the test loss and test accuracy
y_hat =another_vggnet_model.predict(X_test)
test =another_vggnet_model.evaluate(X_test, y_test)
print('Test Loss = ', test[0], 'Test Accuracy = ', test[1])
```

```
6/6 [==============================] - 3s 476ms/step
6/6 [==============================] - 3s 478ms/step - loss: 1.8749 - accuracy: 0.5667
Test Loss =  1.8749417066574097 Test Accuracy =  0.5666666626930237
```

95% - 0.17

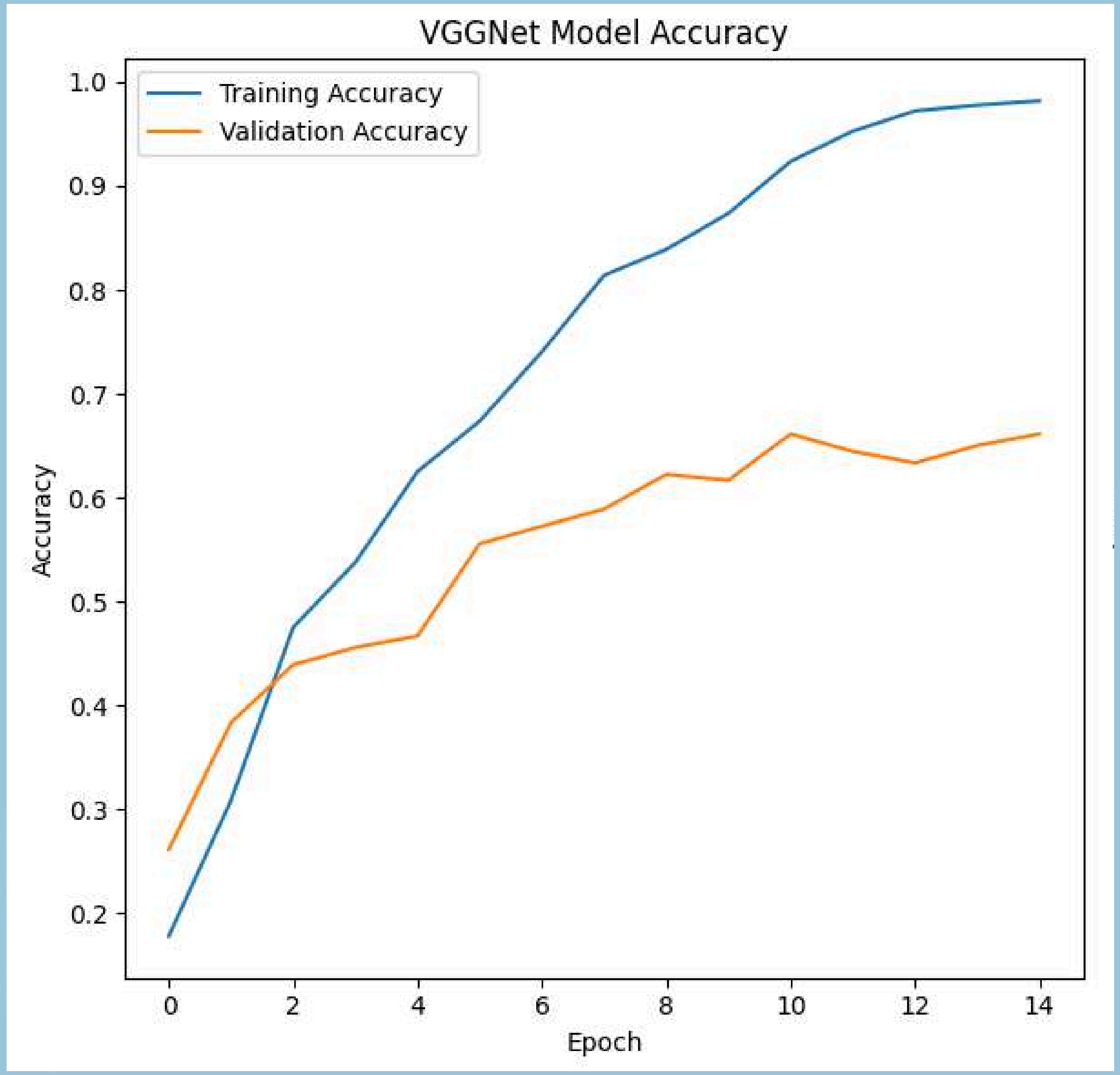
Training Accuracy - Loss

56% - 1.37

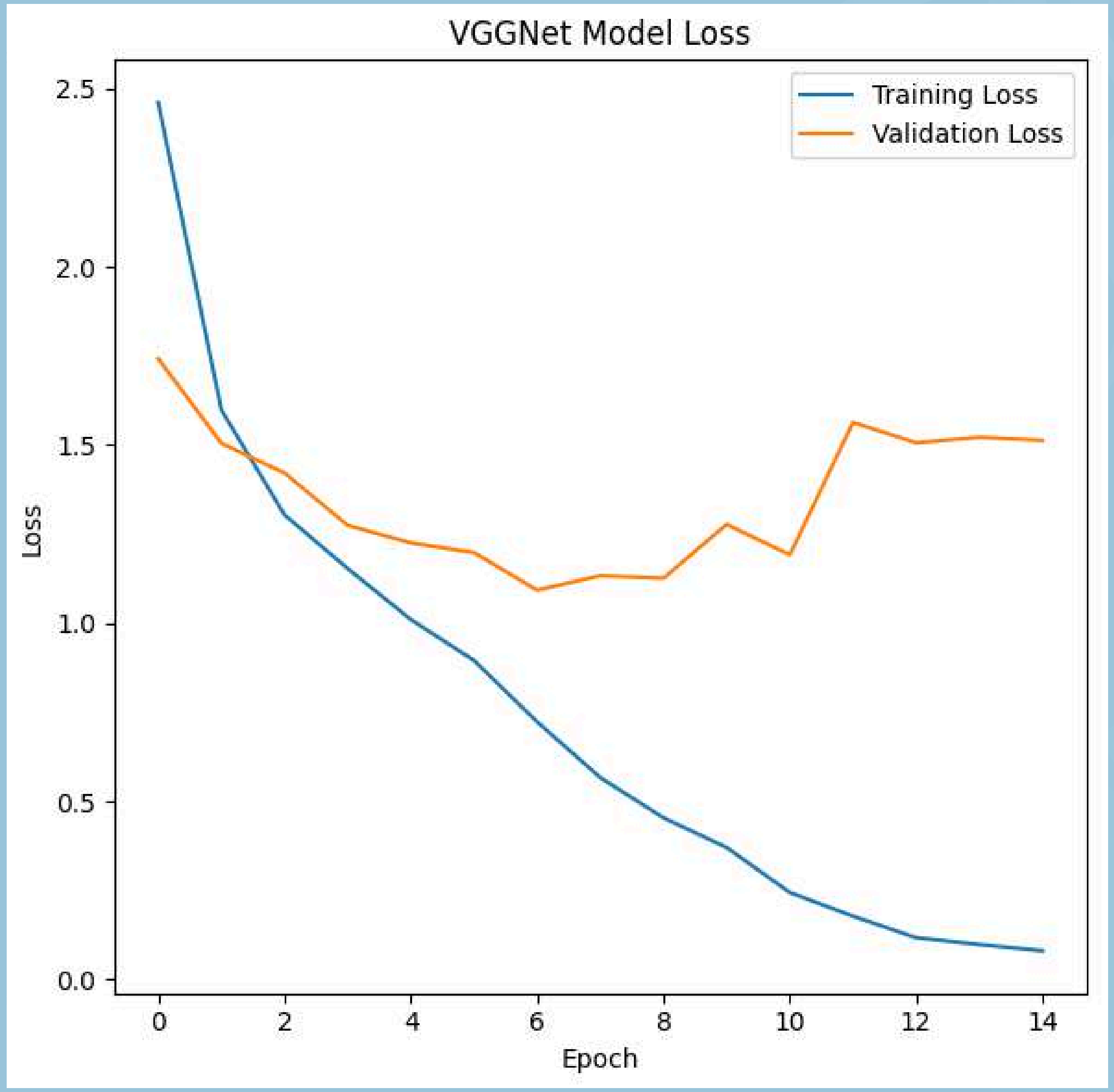
Testing Accuracy - Loss

VGGNet Model

Model Accuracy

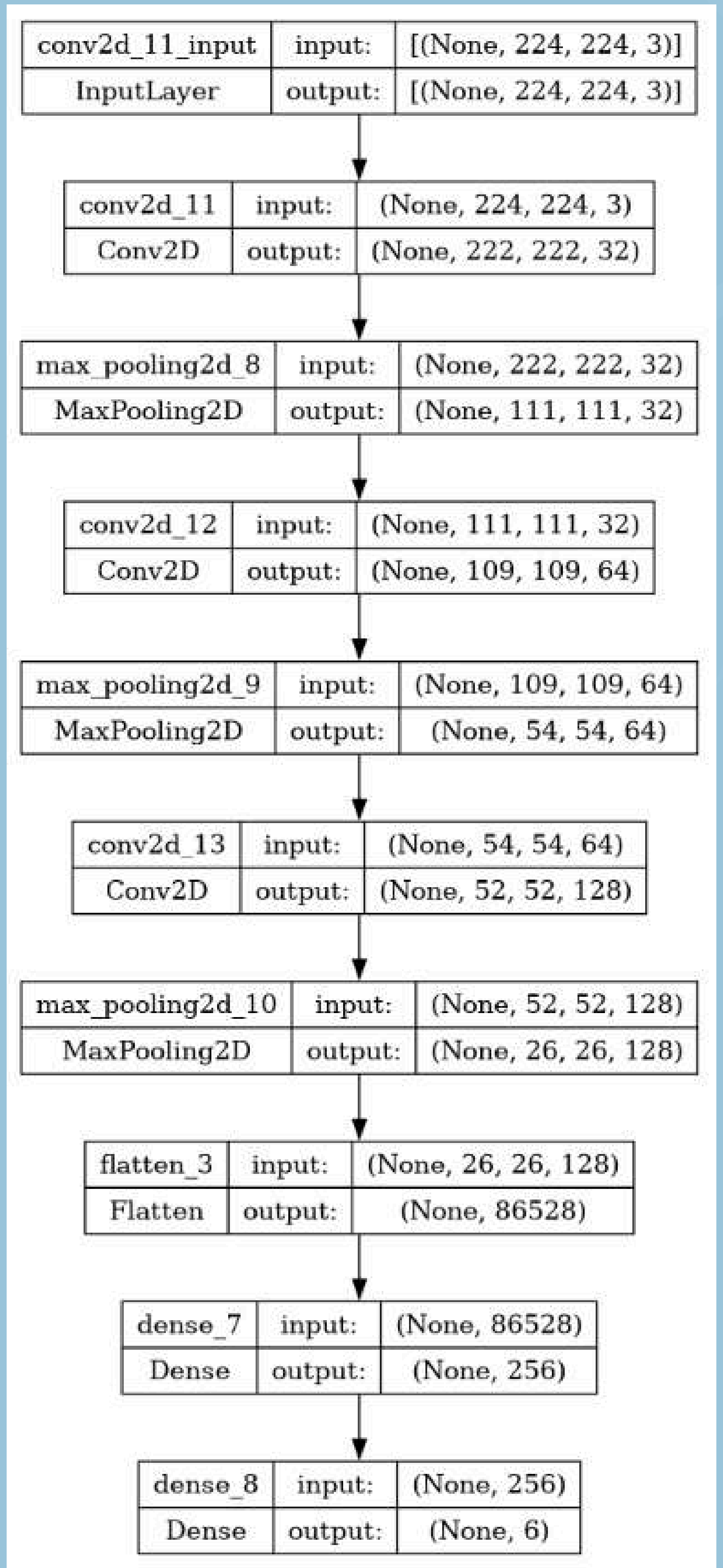


Model Loss

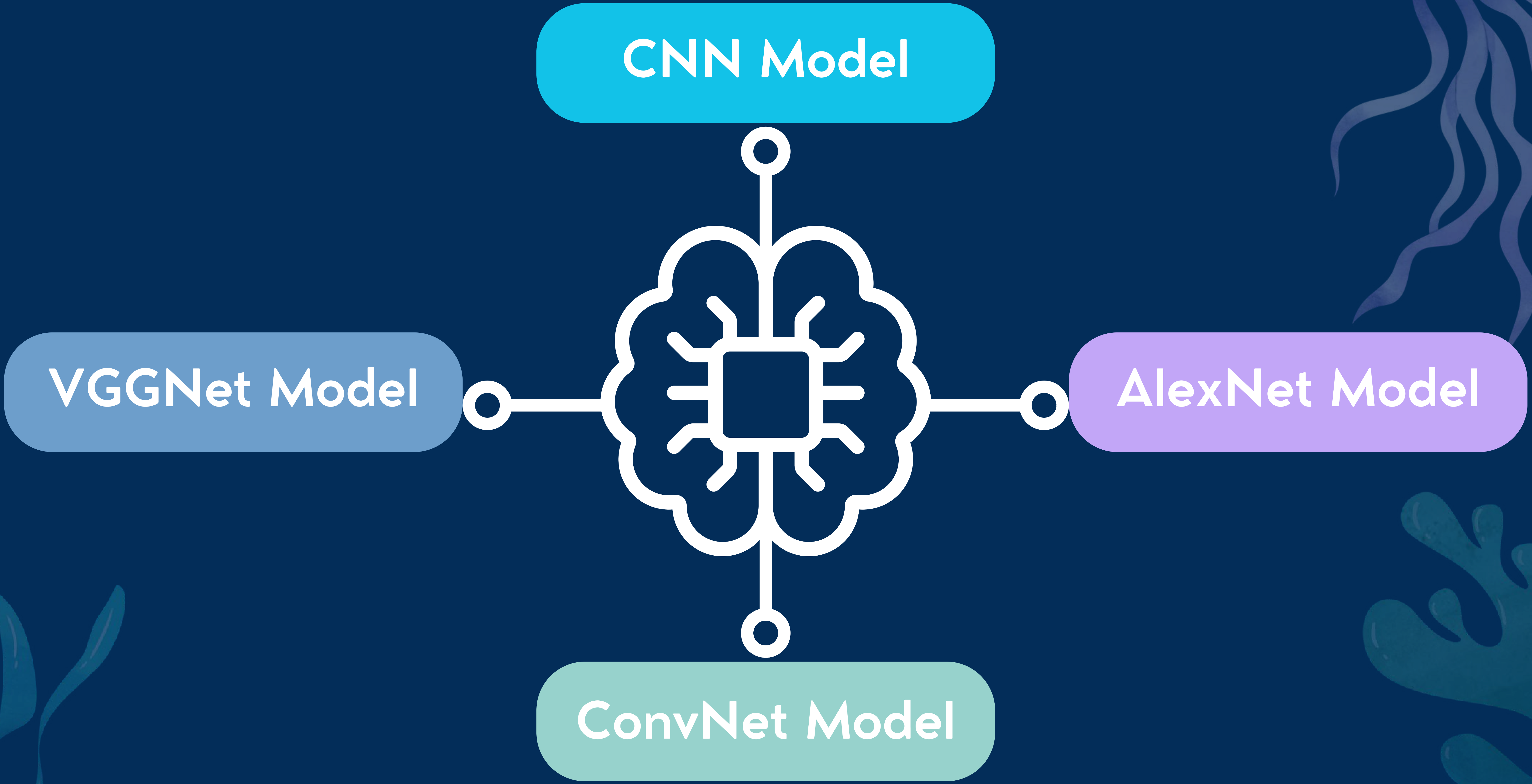


VGGNet Model

Architecture Diagram



OUR MODELS

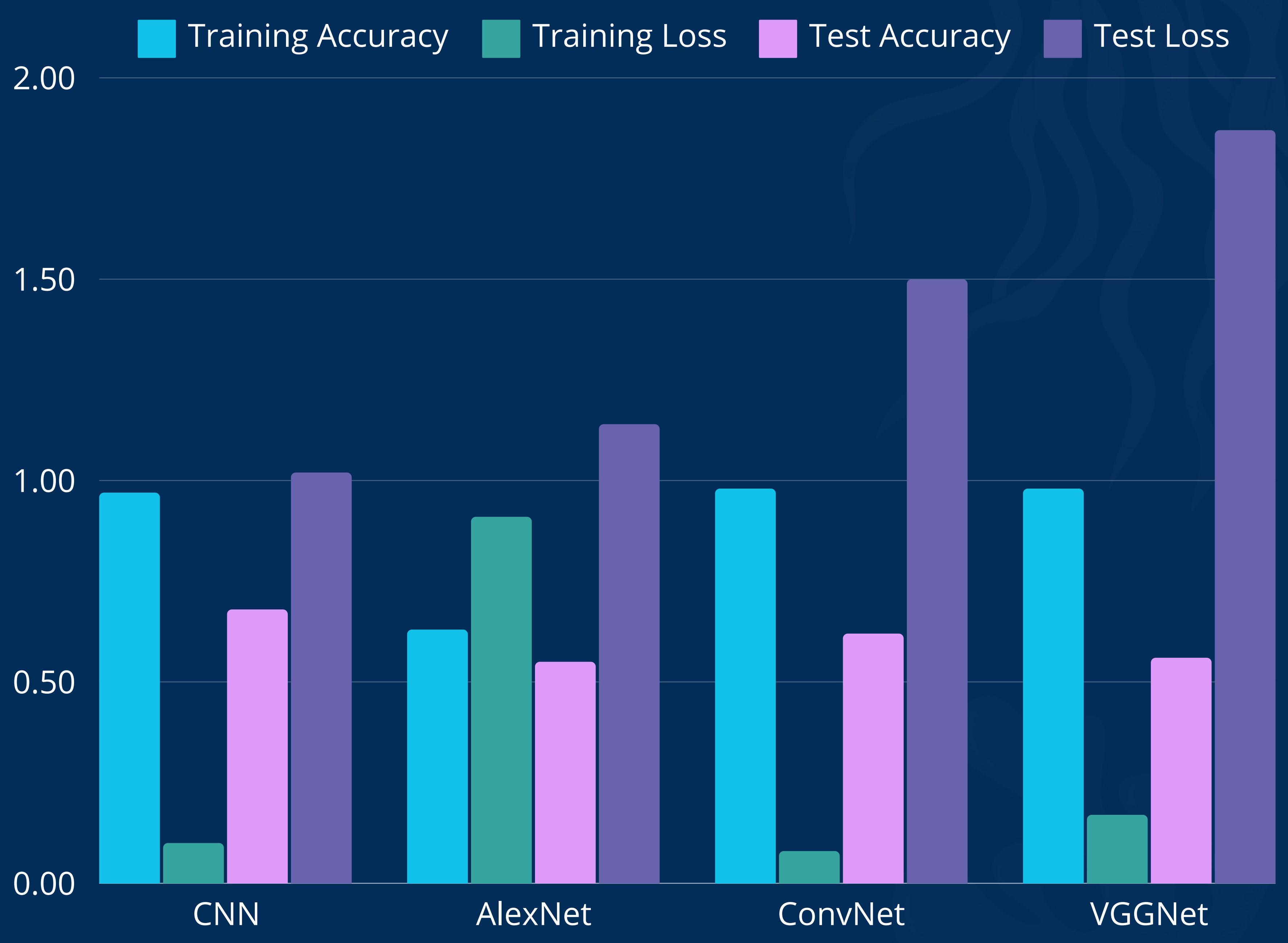


ANALYSIS THE RESULTS

After seeing the results of the previous models, we can conclude:

- The loss rate in the test set is very high.
- The accuracy rate on the test set in all models is less than 80%.
- In some models, there is a large difference between the accuracy rate of the training set and the accuracy rate of the test set. This means that there is overfitting
- The best model among them is the CNN model.

As a result of trying several models, we did not obtain the expected degree of improvement in the accuracy rate, so we will use another method of improvement.



PARAMETERS MODIFICATION

PARAMETERS MODIFICATION

Another method of improving the model we will use is **parameter modification.** After the previous comparison of the models, we will continue with the CNN model because it gave the best performance results.

The parameter we will modify is epochs. This is because it is considered one of the important parameters during deep learning and has many benefits, including:

1. **Model convergence**
2. **Improve the performance**
3. **Dealing with complex data sets**

Because our data is somewhat complicated due to the similarity of the jellyfish's colors to coral reefs, it can help us overcome this problem and obtain the highest accuracy result.

CNN Model After Modification

Define the Model

```
from keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes=6)
y_test = to_categorical(y_test, num_classes=6)
```

```
history = model.fit(train_generator  epochs=100, validation_data=val_generator)
```

We replaced the number of epochs from 20 to 100

CNN Model After Modification

Model Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 224, 224, 16)      208

 max_pooling2d_3 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 112, 112, 32)      2080

 max_pooling2d_4 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 56, 56, 64)        8256

 max_pooling2d_5 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 dropout_2 (Dropout)         (None, 28, 28, 64)        0

 flatten_1 (Flatten)         (None, 50176)             0

 dense_2 (Dense)             (None, 255)               12795135

 dropout_3 (Dropout)         (None, 255)               0

 dense_3 (Dense)             (None, 6)                 1536

=================================================================
Total params: 12807215 (48.86 MB)
Trainable params: 12807215 (48.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

CNN Model After Modification

Accuracy

```python
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
# Find the epoch with the highest validation accuracy
best_epoch = val_accuracy.index(max(val_accuracy)) + 1  # Add 1 because epochs are 1-indexed
print(f"Best Accuracy: {accuracy[best_epoch-1]:.4f} at Epoch {best_epoch}")
print(f"Best Validation Accuracy: {val_accuracy[best_epoch-1]:.4f} at Epoch {best_epoch}")
```

```
Best Accuracy: 0.9681 at Epoch 85
Best Validation Accuracy: 0.9556 at Epoch 85
```

+ Code + Markdown

```python
y_hat = model.predict(X_test)
test = model.evaluate(X_test, y_test)
print('Test Loss = ', test[0], 'Test Accuracy = ', test[1])
```

```
6/6 [==============================] - 1s 187ms/step
6/6 [==============================] - 1s 188ms/step - loss: 0.4609 - accuracy: 0.8778
Test Loss =  0.460921972990036 Test Accuracy =  0.8777777552604675
```

96% - 0.09

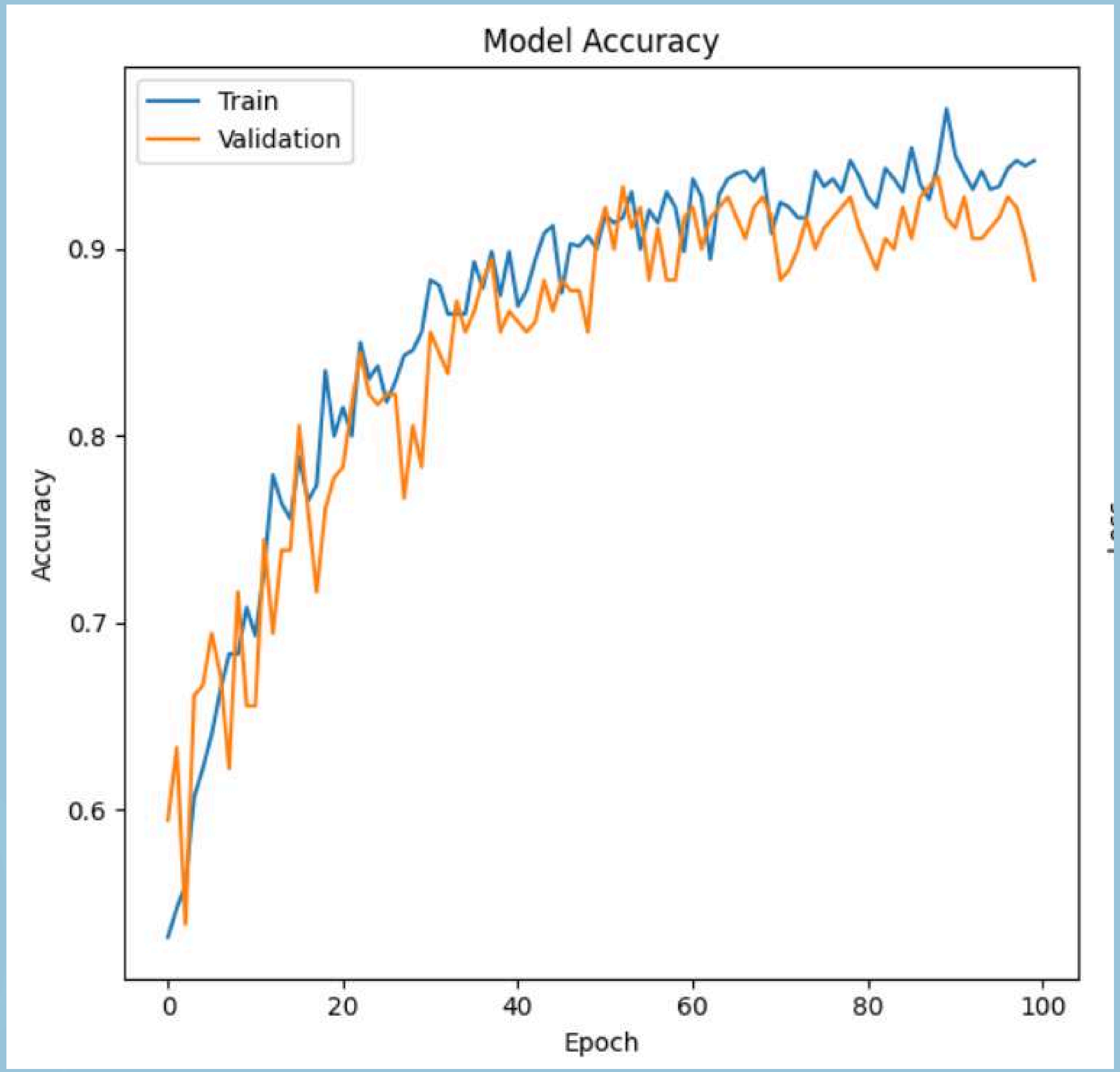
Training Accuracy - Loss

87% - 0.46

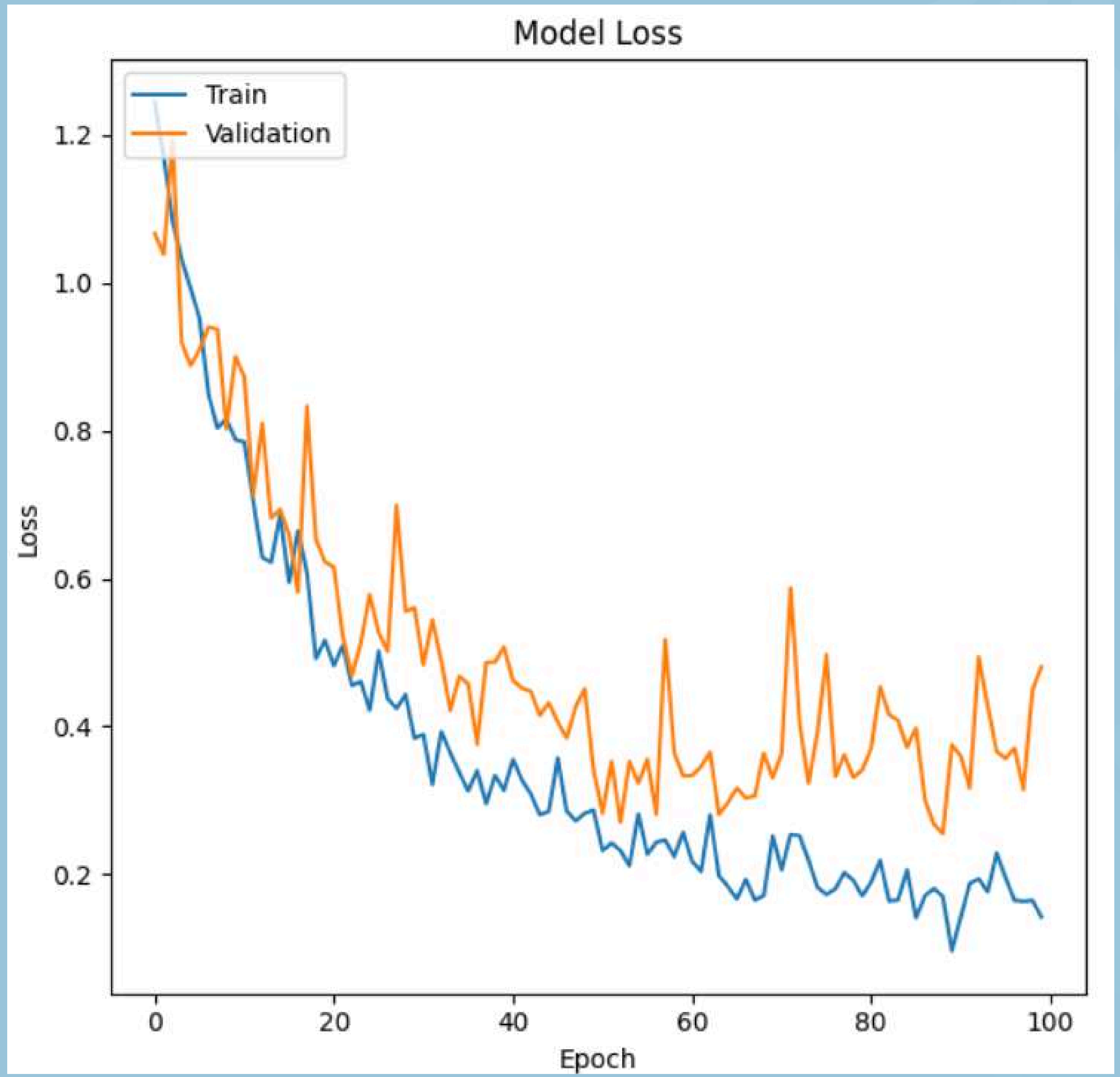
Testing Accuracy - Loss

CNN Model After Modification

Model Accuracy

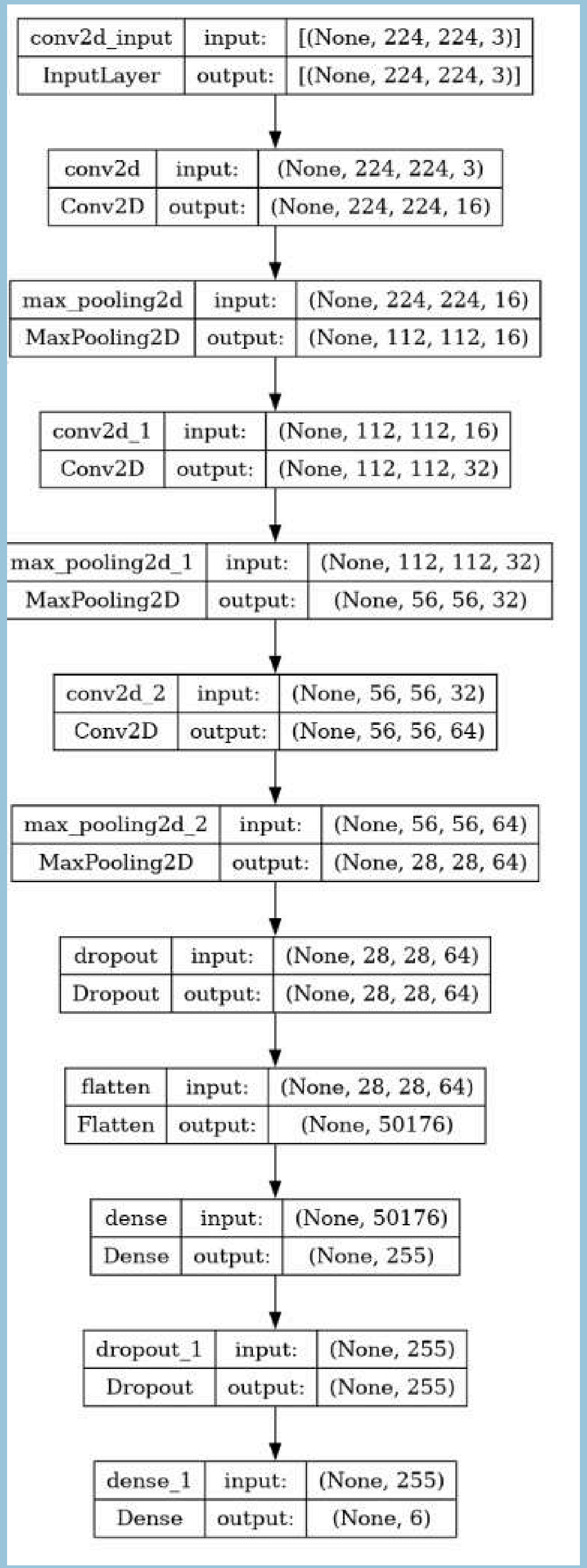


Model Loss



CNN Model After Modification

Architecture Diagram

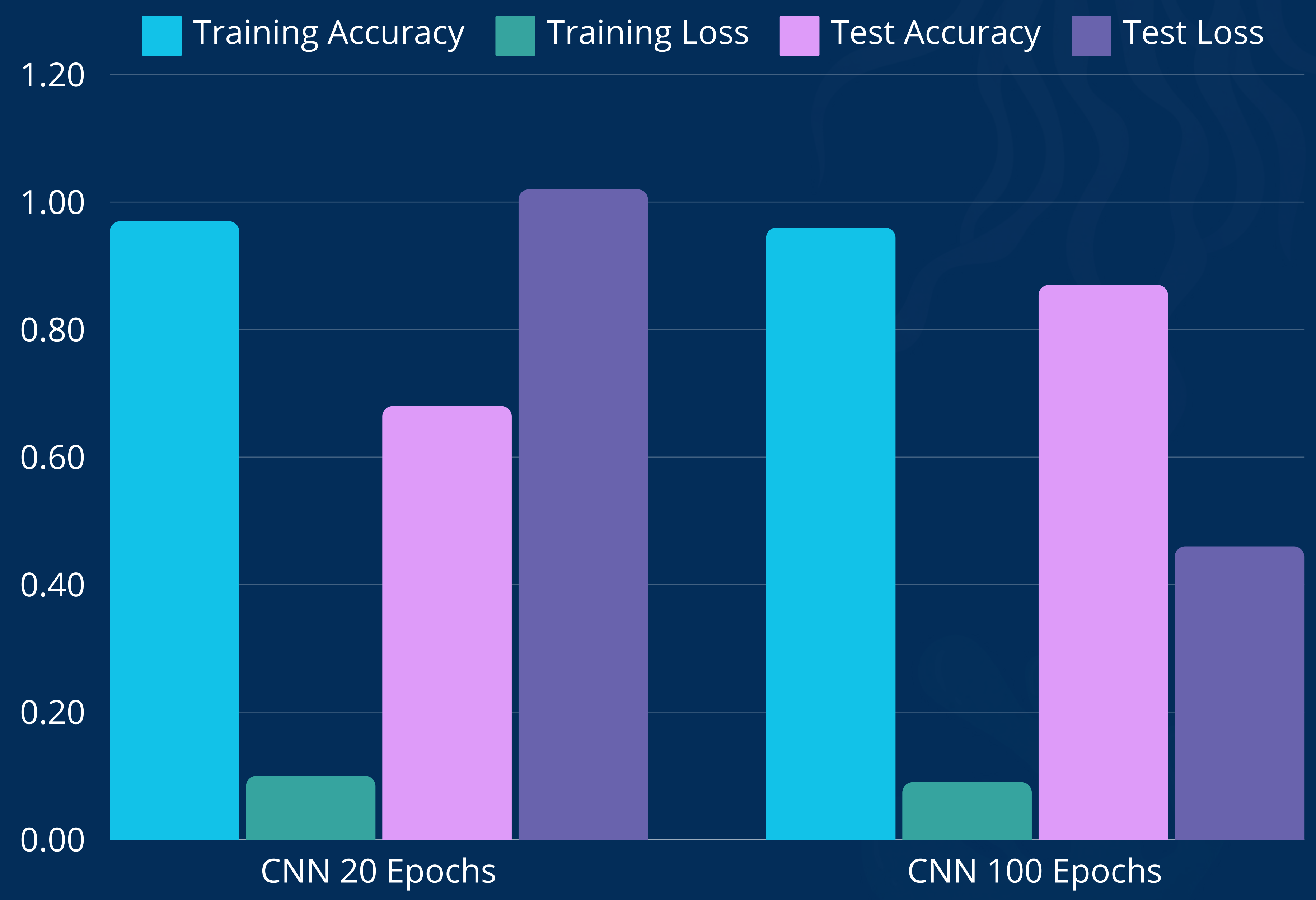


ANALYSIS THE RESULTS

As shown in the chart:
The accuracy rate on the training set increased significantly when we replaced the number of epochs from 20 to 100.
As shown in the chart:

- Accuracy rate of the test set in the old model was: 68%
- Accuracy rate of the test set in the model after modification: 87%

After improving the accuracy rate by changing the number of epochs, we faced a problem, which is that the learning time was very long due to the large number of epochs, and several epochs had a high verification loss, so we tried to search for a solution to this problem.



PARAMETERS MODIFICATION

To solve the previous problem, we added a **performance optimization** code that provides many benefits and best practices in training deep learning models, which include:

- **Reduce the learning rate**
- **Monitor and stop early**
- **Evaluation and visualization**
- **Optimizer selection**

Because our problem is the long learning time due to the large number of epochs and also the high verification loss, this improvement can help us solve this problem.

CNN Model After Modification

```
from keras.callbacks import ReduceLROnPlateau
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1, min_lr=1e-6)

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

history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), callbacks=[lr_scheduler])

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

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=500, epochs=20, callbacks=[lr_scheduler])

y_hat = model.predict(X_test)
test = model.evaluate(X_test, y_test)
print('Test Loss = ', test[0], 'Test Accuracy = ', test[1])

fig = plt.figure(figsize=(20, 8))
for i, idx in enumerate(np.random.choice(X_test.shape[0], size=16, replace=False)):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(X_test[idx]))
    pred_idx = np.argmax(y_hat[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(classes[pred_idx], classes[true_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```

CNN Model After Modification

92%

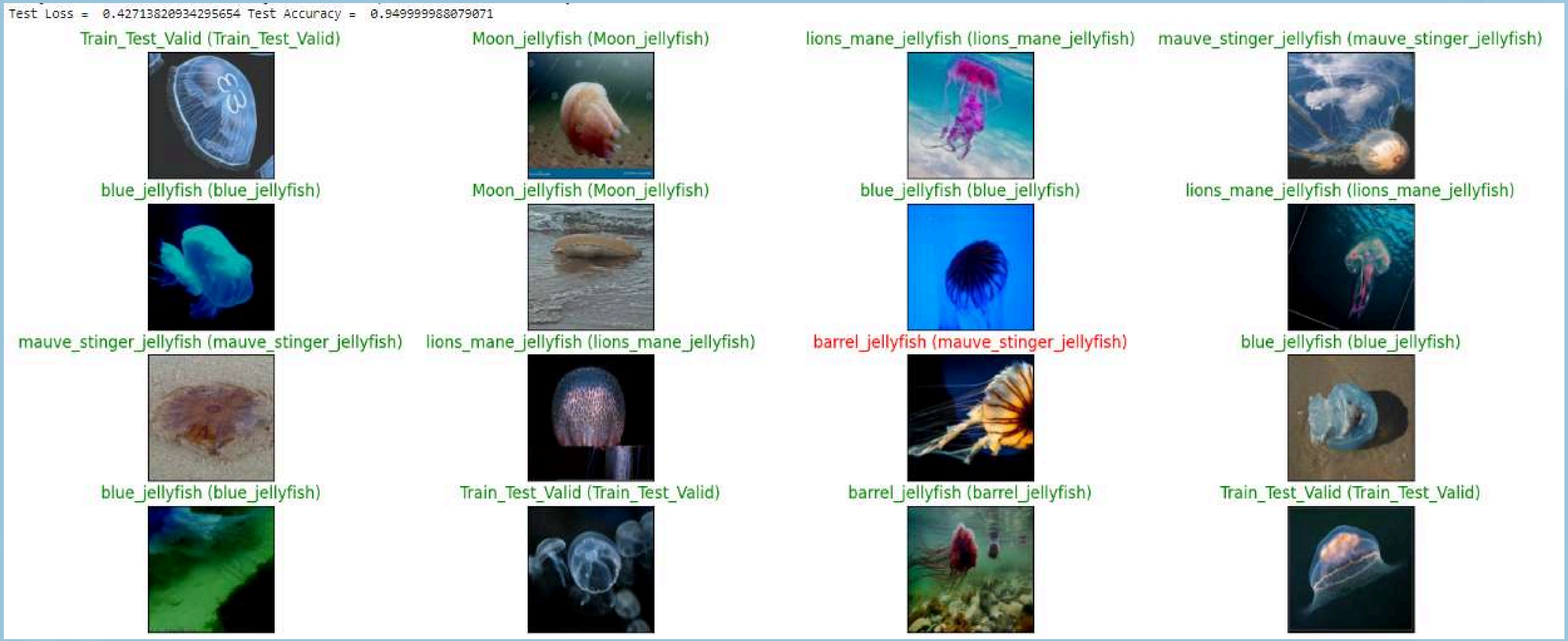
Testing Accuracy

0.37

Testing Loss

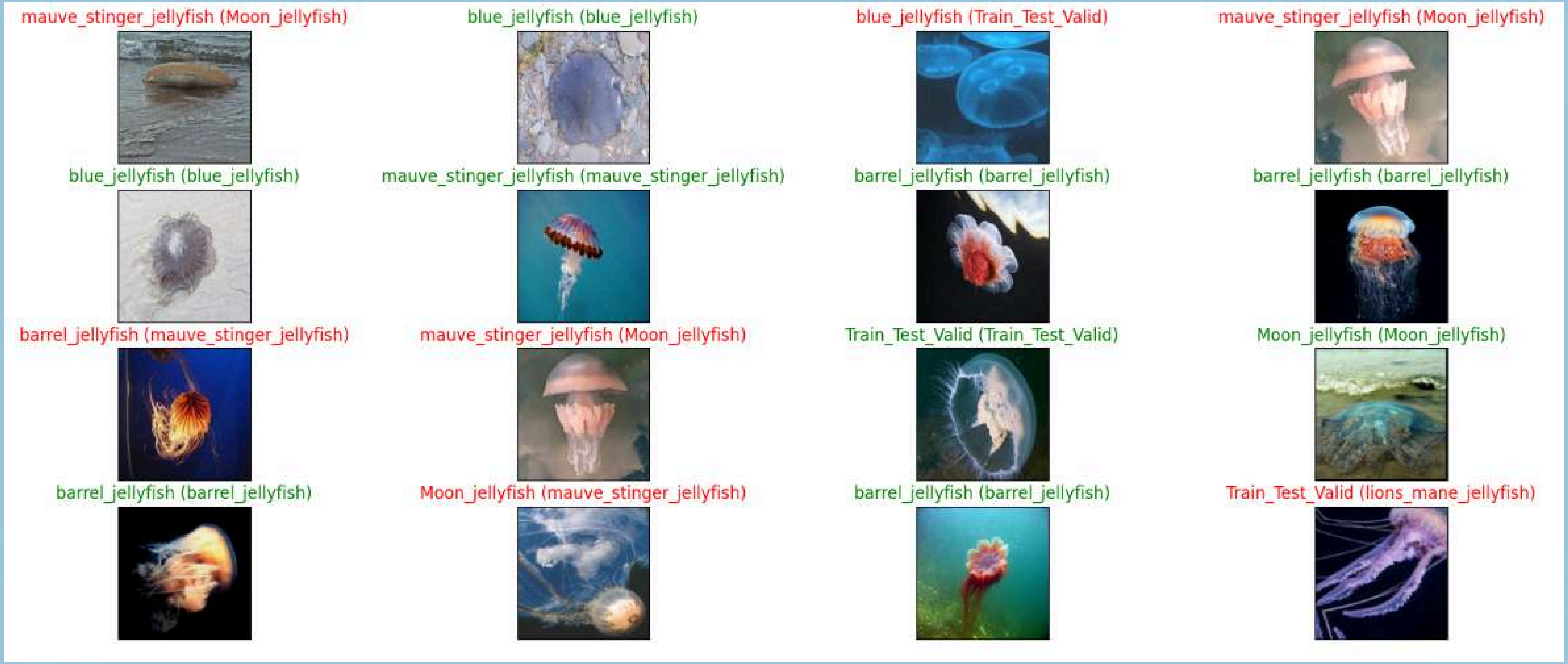
```
Epoch 20/20
23/23 [==============================] - 18s 808ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.3489 - val_accuracy: 0.9333 - lr: 1.0000e-06
Epoch 1/20
2/2 [==============================] - 23s 7s/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.2573 - val_accuracy: 0.9444 - lr: 0.0010
Epoch 2/20
2/2 [==============================] - 17s 6s/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.3046 - val_accuracy: 0.9278 - lr: 0.0010
Epoch 3/20
2/2 [==============================] - 17s 6s/step - loss: 0.0050 - accuracy: 0.9986 - val_loss: 0.3205 - val_accuracy: 0.9333 - lr: 0.0010
Epoch 4/20
2/2 [==============================] - ETA: 0s - loss: 0.0022 - accuracy: 0.9986
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
2/2 [==============================] - 17s 6s/step - loss: 0.0022 - accuracy: 0.9986 - val_loss: 0.4097 - val_accuracy: 0.9222 - lr: 0.0010
Epoch 5/20
2/2 [==============================] - 16s 6s/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.4021 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 6/20
2/2 [==============================] - 17s 7s/step - loss: 5.5192e-04 - accuracy: 1.0000 - val_loss: 0.3975 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 7/20
2/2 [==============================] - ETA: 0s - loss: 0.0018 - accuracy: 1.0000
Epoch 7: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
2/2 [==============================] - 17s 6s/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.3799 - val_accuracy: 0.9278 - lr: 1.0000e-04
Epoch 8/20
2/2 [==============================] - 18s 7s/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.3786 - val_accuracy: 0.9278 - lr: 1.0000e-05
Epoch 9/20
2/2 [==============================] - 17s 7s/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.3770 - val_accuracy: 0.9333 - lr: 1.0000e-05
Epoch 10/20
2/2 [==============================] - ETA: 0s - loss: 0.0066 - accuracy: 0.9986
Epoch 10: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
2/2 [==============================] - 17s 6s/step - loss: 0.0066 - accuracy: 0.9986 - val_loss: 0.3746 - val_accuracy: 0.9278 - lr: 1.0000e-05
Epoch 11/20
2/2 [==============================] - 16s 6s/step - loss: 0.0035 - accuracy: 0.9986 - val_loss: 0.3744 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 12/20
2/2 [==============================] - 17s 6s/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.3742 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 13/20
2/2 [==============================] - ETA: 0s - loss: 0.0011 - accuracy: 1.0000
Epoch 13: ReduceLROnPlateau reducing learning rate to 1e-06.
2/2 [==============================] - 17s 6s/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.3741 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 14/20
2/2 [==============================] - 17s 6s/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.3739 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 15/20
2/2 [==============================] - 17s 7s/step - loss: 5.7488e-04 - accuracy: 1.0000 - val_loss: 0.3737 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 16/20
2/2 [==============================] - 17s 6s/step - loss: 0.0045 - accuracy: 0.9972 - val_loss: 0.3735 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 17/20
2/2 [==============================] - 17s 7s/step - loss: 7.8765e-04 - accuracy: 1.0000 - val_loss: 0.3734 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 18/20
2/2 [==============================] - 17s 6s/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.3733 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 19/20
2/2 [==============================] - 17s 6s/step - loss: 0.0044 - accuracy: 0.9972 - val_loss: 0.3731 - val_accuracy: 0.9278 - lr: 1.0000e-06
Epoch 20/20
2/2 [==============================] - 17s 6s/step - loss: 0.0039 - accuracy: 0.9986 - val_loss: 0.3729 - val_accuracy: 0.9278 - lr: 1.0000e-06
6/6 [==============================] - 1s 190ms/step - loss: 0.3729   accuracy: 0.9278
Test Loss =  0.37286147475242615 Test Accuracy =  0.9277777671813965
```

CNN Model After Modification



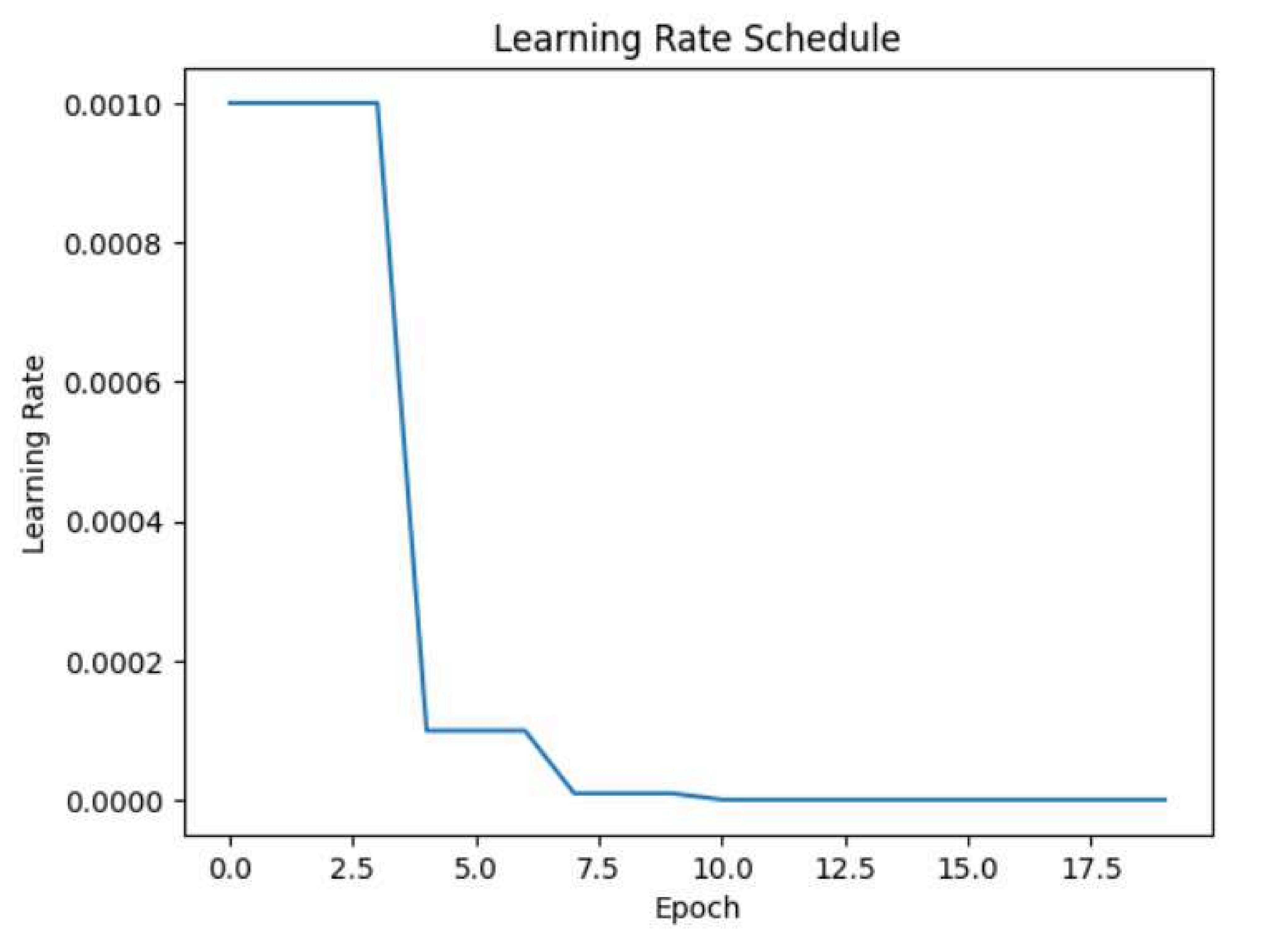
Phase 1

CNN Model Before Modification



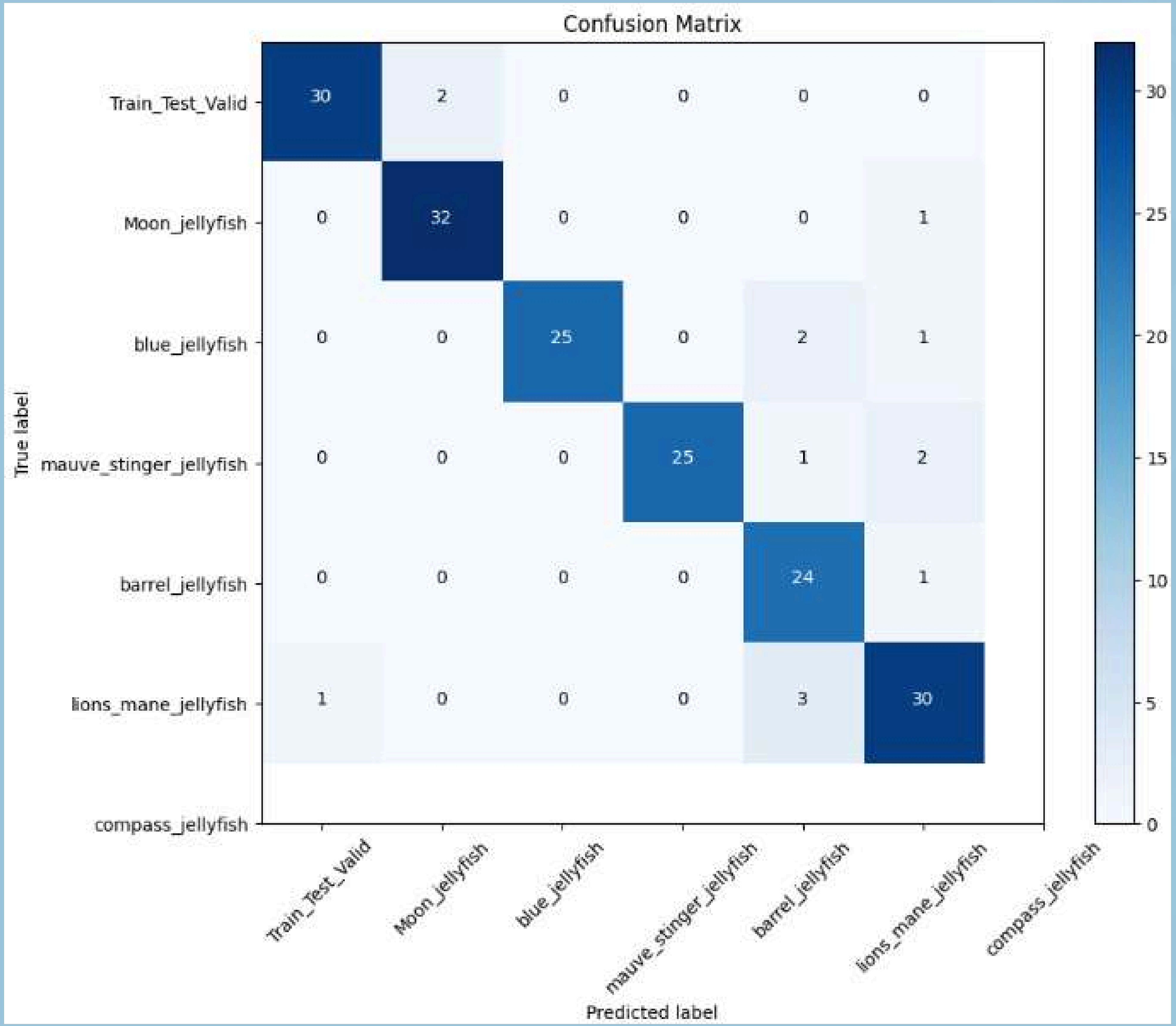
CNN Model After Modification

Learning Rate



CNN Model After Modification

Confusion Matrix



CONCLUSION

Summarize and Analyze Everything

In the preprocessing phase, we augmented the data by:

- **Randomly shift images horizontally and vertically.**
- **Randomly zoom.**
- **Randomly flip images horizontally and vertically.**

Thus we have increased the number of data images.

We used two methods to improve the results of the first phase:

First: Three new models were used, but all of them led to poorly accurate results.

Second: Modifying the parameters:

- We changed the epochs parameter in the CNN model.
- Added improvements to obtain more effective model training, better convergence.
- Improved generalization performance, which leads to more accurate and reliable models.

The accuracy result of our project was improved up to **92%**.

WORK TEAM

01 JOOD ALJOHANI

02 MUNTAHA ALDHAHRI

03 SADEEM ALZAHRANI

04 ASMA HABADI

THANK YOU!