Deep Learning (ANN,CNN,RNN) with Keras - Interview-Questions -Answers ¶

1. What is Keras?

→ High-level deep learning API in Python, runs on TensorFlow, simplifies model building.

2. Difference between ANN, CNN, RNN?

→ ANN → fully connected layers; CNN → convolution for images; RNN → sequential data with memory.

3. What is transfer learning?

→ Using pre-trained models on new tasks to save time and improve performance.

4. What is an epoch in training?

→ One complete pass of the dataset through the network.

5. What is batch size?

→ Number of samples processed before updating model weights.

6. Activation functions in Keras?

→ Sigmoid, ReLU, Tanh, Softmax, Leaky ReLU, etc.

7. Difference between binary and multi-class classification?

→ Binary → 2 classes, output sigmoid; Multi-class → >2 classes, output softmax.

8. Loss functions for classification?

→ Binary: `binary_crossentropy` , Multi-class: `categorical_crossentropy` .

9. Loss functions for regression?

→ `mean_squared_error (MSE)` , `mean_absolute_error (MAE)` .

10. How to calculate trainable parameters in ANN?

```
→ Parameters = input_size * neurons + bias (per layer), sum across la
yers.
```

11. How to calculate parameters in CNN?

→ Conv layer: `(kernel_height * kernel_width * input_channels + 1) * filters`.

12. Pooling layers in CNN?

→ Reduce spatial dimensions: MaxPooling, AveragePooling.

13. Flatten layer in CNN?

→ Converts 2D feature maps to 1D vector for dense layers.

14. RNN basics

→ Handles sequential data; maintains memory via hidden states.

15. LSTM vs GRU

→ LSTM → forget, input, output gates; GRU → combines gates, simpler.

16. Transfer learning workflow

→ Load pre-trained model → freeze layers → add custom layers → train on new dataset.

17. Image size importance in CNN

→ Must match model input; influences output feature map size.

18. Padding in CNN

→ `valid` → no padding, `same` → output same size as input.

19. Stride in CNN

→ Steps the filter moves; controls output size.

20. Dropout layer

→ Prevents overfitting by randomly dropping neurons during training.

21. Early stopping

→ Stops training when validation loss stops improving.

22. Optimizers in Keras

→ SGD, Adam, RMSprop; control learning rate and weight updates.

23. Learning rate

```
→ Step size for updating weights during gradient descent.
```

24. Difference between Sequential and Functional API

```
→ Sequential → linear stack; Functional → complex architectures (mult
i-input/output).
```

25. Callbacks in Keras

```
→ Functions triggered during training (EarlyStopping, ModelCheckpoin
t, ReduceLROnPlateau).
```

26. Build simple ANN for binary classification

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(16, activation='relu', input_shape=(10,)),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
```

27. Train ANN

```
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_spli
t=0.2)
```

28. Build CNN for image classification

```
from keras.layers import Conv2D, MaxPooling2D, Flatten

cnn = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(64,64,3)),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
cnn.compile(optimizer='adam', loss='categorical_crossentropy', metric
s=['accuracy'])
```

29. Calculate CNN parameters

- Conv layer params = `(kernel_h * kernel_w * input_ch + 1) * filters`
- Dense layer params = `(input_units * output_units) + bias`

30. Build simple RNN

```
from keras.layers import SimpleRNN

rnn = Sequential([
    SimpleRNN(32, input_shape=(10,1)),
    Dense(1, activation='sigmoid')
])
rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a
ccuracy'])
```

31. LSTM example

```
from keras.layers import LSTM

lstm = Sequential([
    LSTM(50, input_shape=(20,1)),
    Dense(1, activation='sigmoid')
])
lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
```

32. GRU example

```
from keras.layers import GRU

gru = Sequential([
    GRU(50, input_shape=(20,1)),
    Dense(1, activation='sigmoid')
])
gru.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a
ccuracy'])
```

33. Image preprocessing for CNN

```
from keras.preprocessing.image import ImageDataGenerator
train_gen = ImageDataGenerator(rescale=1./255)
train_data = train_gen.flow_from_directory('train', target_size=(64,6
4), batch_size=32, class_mode='categorical')
```

34. Data augmentation

```
train_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
```

35. Transfer learning with VGG16

```
from keras.applications import VGG16
from keras.layers import Dense, Flatten
from keras.models import Model

base = VGG16(weights='imagenet', include_top=False, input_shape=(224,
224,3))
x = Flatten()(base.output)
x = Dense(128, activation='relu')(x)
output = Dense(10, activation='softmax')(x)
model = Model(inputs=base.input, outputs=output)
for layer in base.layers:
    layer.trainable = False
model.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
```

36. Early stopping

```
from keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', patience=5, restore_best_weigh
ts=True)
model.fit(X_train, y_train, validation_split=0.2, epochs=50, callback
s=[es])
```

37. Dropout example

```
from keras.layers import Dropout
model = Sequential([
    Dense(64, activation='relu', input_shape=(20,)),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

38. Compile for regression

```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=
['mae'])
```

39. Binary classification evaluation

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

40. Multi-class classification evaluation

```
from sklearn.metrics import classification_report
y_pred_classes = model.predict(X_test).argmax(axis=1)
print(classification_report(y_test, y_pred_classes))
```

41. Flatten layer in CNN

```
Flatten()(Conv2D(32, (3,3), activation='relu')(input_layer))
```

42. MaxPooling example

```
MaxPooling2D(pool_size=(2,2))(Conv2D(32, (3,3), activation='relu')(input_layer))
```

43. Sequential API example

```
Sequential([
    Dense(32, activation='relu', input_shape=(10,)),
    Dense(1, activation='sigmoid')
])
```

44. Functional API example

```
from keras.layers import Input
input_layer = Input(shape=(10,))
x = Dense(32, activation='relu')(input_layer)
output = Dense(1, activation='sigmoid')(x)
model = Model(inputs=input_layer, outputs=output)
```

45. Compile with metrics

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

46. Save and load model

```
model.save('model.h5')
from keras.models import load_model
loaded_model = load_model('model.h5')
```

47. Callbacks for LR reduction

```
from keras.callbacks import ReduceLROnPlateau
lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3)
```

48. Custom metric

```
import keras.backend as K
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    return true_positives / (possible_positives + K.epsilon())
```

🔷 Gradient Issues & Batch Normalization – Summary

1️⃣ Vanishing Gradient

- **Definition:** Gradients become extremely small during backpropagation in deep networks.

- **Cause:** Activation functions like sigmoid/tanh squash values → derivatives < 1 → gradients shrink across layers.
- **Effect:** Early layers learn very slowly; network stops improving.
- **Solution:**
  - Use **ReLU** or variants (Leaky ReLU, ELU) instead of sigmoid/tanh.
  - Apply **Batch Normalization**.
  - Use **residual connections (ResNet)**.

2️⃣ Exploding Gradient

- **Definition:** Gradients become excessively large during backpropagation.
- **Cause:** Weights accumulate large updates → gradient magnitudes grow exponentially.
- **Effect:** Training becomes unstable; weights may become NaN or overflow.
- **Solution:**
  - Apply **Gradient Clipping** (limit gradient values).
  - Proper **weight initialization** (He, Xavier).
  - Reduce **learning rate**.

3️⃣ Batch Normalization (BatchNorm)

- **Definition:** Normalizes layer inputs to have zero mean and unit variance for each mini-batch.
- **Formula:**
  [\hat{x} = \frac{x - \text{mean}*{batch}}{\sqrt{\text{var}*{batch} + \epsilon}}]
  [y = \gamma \hat{x} + \beta]
- **Effect:**
  - Reduces **internal covariate shift**.
  - Stabilizes training and allows **higher learning rates**.
  - Mitigates **vanishing/exploding gradients**.
  - Acts as a mild regularizer (reduces need for Dropout).

🔹 Summary Table

Problem	Cause	Effect	Solution
Vanishing Gradient	Sigmoid/tanh → small derivatives	Slow learning in early layers	ReLU, BatchNorm, Residual connections
Exploding Gradient	Large weight updates	Unstable training, NaN weights	Gradient clipping, proper init, lower LR
Batch Normalization	Internal covariate shift	Training instability	Normalize batch inputs, learn γ & β params